Lab EmbSw1

Michael Schmid

July 23, 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 **11**
2.1 Aufgabe 1: Klasse für die Speicherung von Messwerten . . . 11
2.2 Aufgabe 2: Komplexitätsbetrachtungen . . . 15
2.3 Aufgabe 3: Implementation eines dynamischen Stacks mit Hilfe einer verketteten Liste . . . 16

3 Lab 3 Sortieralgorithmen und Komplexitätstheorie **18**
3.1 Aufgabe 1: Zeitmessung bei Fibonacci-Implementationen . . . 18
3.2 Aufgabe 2: Klasse für Stoppuhr . . . 20
3.3 Aufgabe 3: Komplexität . . . 21
3.4 Aufgabe 4: Sortieralgorithmen . . . 22

4 Lab 4 Zufallszahlengeneratoren **25**
4.1 Aufgabe 1: Verteilung der Zufallswerte durch `rand()` . . . 25
4.2 Aufgabe 2: Güte von Zufallszahlengeneratoren . . . 26
4.3 Aufgabe 3: Nicht ideale Münze (biased coin) . . . 30

5 Lab 5 Dinierende Philosophen **31**
5.1 Aufgabe 1: Untersuchung von Deadlocks . . . 32
5.2 Aufgabe 2: Implementation der dinierenden Philosophen . . . 33

6 Lab 6 CRC - Berechnung und -Implementation in C++ **37**
6.1 Aufgabe 1: CRC-8 Berechnungsbeispiel (Papierübung) . . . 37
6.2 Aufgabe 2: Verifikation empfangener Daten mittels CRC . . . 38
6.3 Aufgabe 3: C++ - Implementation . . . 39
6.4 Aufgabe 4: C++ - Implementation mit einer Lookup Tabelle . . . 40
6.5 Aufgabe 5: C++ - Implementation mit einer Lookup Tabelle . . . 42

7 Lab 7 inline und Bitfelder **44**
7.1 Aufgabe 1: inline-Methoden . . . 44
7.2 Aufgabe 2: Bitfelder . . . 48
7.3 Aufgabe 3: Bitmasken . . . 50

8 Lab 8 Digitale Ableitung und Pulsdetektion auf dem EmbSW-Roboter **51**
8.1 Aufgabe 1: Implementation des FIR-Filters 52
8.2 Aufgabe 2: Implementation der Pulsdetektion 57

9 Lab 9 Code Bloat bei Templates, Code Hoisting **58**
9.1 Aufgabe 1: Untersuchung von Assemblercode bei Templates 58
9.2 Aufgabe 2: Verhindern von Code Bloat durch Code Hoisting 61

10 Lab 10 Dynamic Memory Management **64**
10.1 Aufgabe 1: Fixed-size Pool 64
10.2 Aufgabe 2: Block Allocator 67

11 Lab 11 C++ and ROMability, Hardware Abstraction Layer (HAL) **69**
11.1 Aufgabe 1: C++ and ROMability, Hardware Abstraction Layer (HAL) 69
11.2 Aufgabe 2: Ohne Hardware Abstraction Layer 75
11.3 Aufgabe 3: Hardware Abstraction Layer in C 76
11.4 Aufgabe 4: Hardware Abstraction Layer in C++ 81
11.5 Aufgabe 5: Hardware Abstraction Layer in C++ mit Memory-mapped IO (MMIO) 82

12 Lab 12 Zufallszahlengeneratoren **86**
12.1 Aufgabe 1: Wahrscheinlichkeitsverteilungen 86

13 Lab 13 Security **88**
13.1 Aufgabe 1: Hashing 88
13.2 Aufgabe 1: Verschlüsselten Text entschlüsseln 89

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

```
/*
 * SList.h
 *
 * Singly-linked list for doubles
 *
 *  Created on: 12.02.2016
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_

class SList
{
  public:
    SList();
    ~SList();

    void insertAt(int pos, double val);
    // inserts element at (after) position pos (0: at head)

    void deleteAt(int pos);
    // deletes element at position pos (>0)

    int search(double val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    bool isEmpty() const;
    // returns true if list is empty, else false

    int getNumber() const;
    // returns number of elements

    double getValue(int pos) const;
    // returns value at position pos

    void setValue(int pos, double val);
    // sets value val at position pos

    void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        double value;
        Node* next;
    };
    Node* pHead;  // ptr to head of list
    int nr;       // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

#endif /* SLIST_H_ */
```

```
/*
 * SList.cpp
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#include <cassert>
#include <iostream>
#include "SList.h"
using namespace std;

SList::SList() :
    pHead(nullptr), nr(0)
{
}

SList::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

void SList::insertAt(int pos, double val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else  // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  ++nr;
}

void SList::deleteAt(int pos)
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead;  // cursor
  Node* pDel = p;   // node to be deleted
  if (pos == 1)  // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  --nr;
}

int SList::search(double val) const
{
  Node* p = pHead;  // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0;  // not found
}

bool SList::isEmpty() const
{
  return nr == 0;
}

int SList::getNumber() const
{
  return nr;
}

double SList::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

void SList::setValue(int pos, double val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

void SList::print() const
{
  cout << "--------------------" << endl;
  cout << "Number of elements: " << nr << endl;
  cout << "Content of list:" << endl;
  for (Node* p = pHead; p; p = p->next)
    cout << p->value << " ";
  cout << endl;
}

SList::Node* SList::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead;  // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
```

```cpp
//============================================================
// Name        : Lists.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : List implementations
//============================================================

#include <iostream>
#include "SList.h"
#include "DList.h"
using namespace std;

int main()
{
  SList s;
  cout << "Singly-linked list" << endl;
  s.print();

  s.insertAt(0, 1.23);
  s.insertAt(0, -34.6);
  s.insertAt(0, 0.4);
  s.print();

  s.insertAt(3, 7.97);
  s.insertAt(1, 88.9);
  s.print();

  s.deleteAt(2);
  s.print();
  cout << "-----------------------------------" << endl << endl;

  DList d;
  cout << "Doubly-linked list" << endl;
  d.print();

  d.insertAt(0, 1.23);
  d.insertAt(0, -34.6);
  d.insertAt(0, 0.4);
  d.print();

  d.insertAt(3, 7.97);
  d.insertAt(1, 88.9);
  d.print();

  d.deleteAt(5);
  d.insertAt(4, 44.5);
  d.print();

  return 0;
}
```

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

```cpp
/*
 * SList.h
 *
 * Singly-linked list for type T
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_

#include <cassert>
#include <iostream>

template<typename T>
class SList
{
  public:
    SList() :
      pHead(nullptr), nr(0)
    {

    }

    ~SList();

    void insertAt(int pos, const T& val);
    // inserts element at (after) position pos (0: at head)

    void deleteAt(int pos);
    // deletes element at position pos (>0)

    int search(const T& val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    const T& getValue(int pos) const;
    // returns value at position pos

    void setValue(int pos, const T& val);
    // sets value val at position pos

    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }

    int getNumber() const
    // returns number of elements
    {
      return nr;
    }

    void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        T value;
        Node* next;
    };
    Node* pHead; // ptr to head of list
    int nr; // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

template<typename T>
SList<T>::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void SList<T>::insertAt(int pos, const T& val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  ++nr;
}

template<typename T>
void SList<T>::deleteAt(int pos)
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
```

```
  delete pDel;
  --nr;
}

template<typename T>
int SList<T>::search(const T& val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
const T& SList<T>::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

template<typename T>
void SList<T>::setValue(int pos, const T& val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

template<typename T>
void SList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << nr << std::endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template<typename T>
typename SList<T>::Node* SList<T>::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}

#endif /* SLIST_H_ */
```

```
//
      ============================================================================
// Name        : ListTest.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : List implementations
//
      ============================================================================


#include <iostream>
#include "SList.h"
using namespace std;

int main()
{
  SList<double> s;
  cout << "Singly linked list" << endl;
  s.print();

  s.insertAt(0, 1.23);
  s.insertAt(0, -34.6);
  s.insertAt(0, 0.4);
  s.print();

  s.insertAt(3, 7.97);
  s.insertAt(1, 88.9);
  s.print();

  s.deleteAt(2);
  s.print();

  return 0;
}
```

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

```
/*
 * DList.h
 * Doubly linked list for doubles
 *
 *  Created on: 12.02.2016
 *      Author: rbondere
 */

#ifndef DLIST_H_
#define DLIST_H_

class DList
{
  public:
    DList();
    ~DList();

    void insertAt(int pos, double val);
    // inserts element at (after) position pos (0: at head)

    void deleteAt(int pos);
    // deletes element at position pos (>0)

    int search(double val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    bool isEmpty() const;
    // returns true if list is empty, else false

    int getNumber() const;
    // returns number of elements

    double getValue(int pos) const;
    // returns value at position pos

    void setValue(int pos, double val);
    // sets value val at position pos

    void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        double value;
        Node* next;
        Node* prev;
    };
    Node* pHead;  // ptr to head of list
    int nr;       // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

#endif /* DLIST_H_ */
```

```cpp
/*
 * DList.cpp
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#include <cassert>
#include <iostream>
#include "DList.h"
using namespace std;

DList::DList() :
    pHead(nullptr), nr(0)
{
}

DList::~DList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

void DList::insertAt(int pos, double val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    pEl->prev = p;
    p->next = pEl;
  }
  else  // insert at head
  {
    pEl->next = pHead;
    pEl->prev = nullptr;
    pHead = pEl;
  }
  if (pEl->next != nullptr)  // not last element in list
    pEl->next->prev = pEl;
  ++nr;
}

void DList::deleteAt(int pos)
{
  assert(pos > 0 && pos <= nr);
  Node* pDel = nodePtr(pos);   // node to be deleted
  assert(pDel != nullptr);
  if (pos == 1)  // first element
    pHead = pHead->next;
  else
  {
    pDel->prev->next = pDel->next;
  }
  if (pDel->next != nullptr)  // not last element in list
    pDel->next->prev = pDel->prev;
  delete pDel;
  --nr;
}

int DList::search(double val) const
{
  Node* p = pHead;  // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0;  // not found
}

bool DList::isEmpty() const
{
  return nr == 0;
}

int DList::getNumber() const
{
  return nr;
}

double DList::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

void DList::setValue(int pos, double val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

void DList::print() const
{
  cout << "--------------------" << endl;
  cout << "Number of elements: " << nr << endl;
  cout << "Content of list:" << endl;
  for (Node* p = pHead; p; p = p->next)
    cout << p->value << " ";
  cout << endl;
}

DList::Node* DList::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead;  // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
```

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

```cpp
/*
 * DList.h
 * Double linked list for doubles
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#ifndef DLIST_H_
#define DLIST_H_
#include "List.h"

class DList: public List
{
  public:
    DList();
    virtual ~DList();

    virtual void insertAt(int pos, double val);
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos);
    // deletes element at position pos (>0)

    virtual int search(double val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    int getPrev() const;
    // returns position of previous element (0: no previous, i.e. at head
        )

    virtual double getValue(int pos) const;
    // returns value at position pos

    virtual void setValue(int pos, double val);
    // sets value val at position pos

    virtual void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        double value;
```

```
        Node* next;
        Node* prev;
    };
    Node* pHead; // ptr to head of list

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

#endif /* DLIST_H_ */
```

```
/*
 * DList.cpp
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#include <cassert>
#include <iostream>
#include "DList.h"
using namespace std;

DList::DList() :
    List(), pHead(0)
{
}

DList::~DList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

void DList::insertAt(int pos, double val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    pEl->prev = p;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pEl->prev = nullptr;
    pHead = pEl;
  }
  if (pEl->next != nullptr) // not last element in list
    pEl->next->prev = pEl;
  setNumber(getNumber() + 1);
}

void DList::deleteAt(int pos)
{
  assert(pos > 0 && pos <= getNumber());
  Node* pDel = nodePtr(pos); // node to be deleted
  assert(pDel != nullptr);
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    pDel->prev->next = pDel->next;
  }
  if (pDel->next != nullptr) // not last element in list
    pDel->next->prev = pDel->prev;
  delete pDel;
  setNumber(getNumber() - 1);
}

int DList::search(double val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

double DList::getValue(int pos) const
{
  assert(pos > 0 && pos <= getNumber());
  return nodePtr(pos)->value;
}

void DList::setValue(int pos, double val)
{
  assert(pos > 0 && pos <= getNumber());
  nodePtr(pos)->value = val;
}

void DList::print() const
{
  cout << "--------------------" << endl;
  cout << "Number of elements: " << getNumber() << endl;
  cout << "Content of list:" << endl;
  for (Node* p = pHead; p; p = p->next)
    cout << p->value << " ";
  cout << endl;
}

DList::Node* DList::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= getNumber());
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
```

```
/*
 * SList.h
 *
 * Singly-linked list for doubles
 *
 *  Created on: 16.02.2016
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_
#include "List.h"

class SList: public List
{
  public:
    SList();
    virtual ~SList();

    virtual void insertAt(int pos, double val);
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos);
    // deletes element at position pos (>0)

    virtual int search(double val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    virtual double getValue(int pos) const;
    // returns value at position pos

    virtual void setValue(int pos, double val);
    // sets value val at position pos

    virtual void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        double value;
        Node* next;
    };
    Node* pHead; // ptr to head of list

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

#endif /* SLIST_H_ */
```

```cpp
/*
 * SList.cpp
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#include <cassert>
#include <iostream>
#include "SList.h"
using namespace std;

SList::SList() :
    List(), pHead(nullptr)
{
}

SList::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

void SList::insertAt(int pos, double val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  setNumber(getNumber() + 1);
}

void SList::deleteAt(int pos)
{
  assert(pos > 0 && pos <= getNumber());
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  setNumber(getNumber() - 1);
}

int SList::search(double val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

double SList::getValue(int pos) const
{
  assert(pos > 0 && pos <= getNumber());
  return nodePtr(pos)->value;
}

void SList::setValue(int pos, double val)
{
  assert(pos > 0 && pos <= getNumber());
  nodePtr(pos)->value = val;
}

void SList::print() const
{
  cout << "--------------------" << endl;
  cout << "Number of elements: " << getNumber() << endl;
  cout << "Content of list:" << endl;
  for (Node* p = pHead; p; p = p->next)
    cout << p->value << " ";
  cout << endl;
}

SList::Node* SList::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= getNumber());
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
```

```cpp
//
      ============================================================================
// Name        : Lists.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : List implementations
//
      ============================================================================


#include <iostream>
#include "SList.h"
#include "DList.h"
using namespace std;

int main()
{
  SList s;
  cout << "Singly linked list" << endl;
  s.print();

  s.insertAt(0, 1.23);
  s.insertAt(0, -34.6);
  s.insertAt(0, 0.4);
  s.print();

  s.insertAt(3, 7.97);
  s.insertAt(1, 88.9);
  s.print();

  s.deleteAt(2);
  s.print();
  cout << "-------------------------------" << endl;

  DList d;
  cout << "Doubly linked list" << endl;
  d.print();

  d.insertAt(0, 1.23);
  d.insertAt(0, -34.6);
  d.insertAt(0, 0.4);
  d.print();

  d.insertAt(3, 7.97);
  d.insertAt(1, 88.9);
  d.print();

  d.deleteAt(5);
  d.insertAt(4, 44.5);
  d.print();

  return 0;
}
```

```cpp
/*
 * List.h
 *
 *  Created on: 15.02.2016
 *      Author: rbondere
 */

#ifndef LIST_H_
#define LIST_H_


class List
{
  public:
    List() :
        nr(0)
    {
    }
```

```
    virtual ~List()
    {
    }

    virtual void insertAt(int pos, double val) = 0;
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos) = 0;
    // deletes element at position pos (>0)

    virtual int search(double val) const = 0;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    int getNumber() const
    // returns number of elements
    {
      return nr;
    }

    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }

    virtual double getValue(int pos) const = 0;
    // returns value at position pos

    virtual void setValue(int pos, double val) = 0;
    // sets value val at position pos

    virtual void print() const = 0;
    // prints content of list to console

  protected:
    void setNumber(int nr)
    // sets number of elements
    {
      this->nr = nr;
    }

  private:
    int nr; // number of Elements
};

#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

```
/*
 * DList.h
 * Double linked list for type T
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#ifndef DLIST_H_
#define DLIST_H_

#include <cassert>
#include <iostream>

#include "List.h"

template<typename T>
class DList: public List<T>
{
  public:
    DList() :
      List<T>(), pHead(nullptr)
    {

    }
    virtual ~DList();

    virtual void insertAt(int pos, const T& val);
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos);
    // deletes element at position pos (>0)

    virtual int search(const T& val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    virtual const T& getValue(int pos) const;
    // returns value at position pos

    virtual void setValue(int pos, const T& val);
    // sets value val at position pos

    virtual void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        T value;
        Node* next;
        Node* prev;
    };
    Node* pHead; // ptr to head of list
    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

template<typename T>
DList<T>::~DList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void DList<T>::insertAt(int pos, const T& val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    pEl->prev = p;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pEl->prev = nullptr;
    pHead = pEl;
  }
  if (pEl->next != nullptr) // not last element in list
    pEl->next->prev = pEl;
  List<T>::setNumber(List<T>::getNumber() + 1);
}

template<typename T>
void DList<T>::deleteAt(int pos)
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  Node* pDel = nodePtr(pos); // node to be deleted
  assert(pDel != nullptr);
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    pDel->prev->next = pDel->next;
  }
  if (pDel->next != nullptr) // not last element in list
    pDel->next->prev = pDel->prev;
  delete pDel;
  List<T>::setNumber(List<T>::getNumber() - 1);
}

template<typename T>
int DList<T>::search(const T& val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
const T& DList<T>::getValue(int pos) const
{
```

```
  assert(pos > 0 && pos <= List<T>::getNumber());
  return nodePtr(pos)->value;
}

template<typename T>
void DList<T>::setValue(int pos, const T& val)
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  nodePtr(pos)->value = val;
}

template<typename T>
void DList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << List<T>::getNumber() << std::
      endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template <typename T>
typename DList<T>::Node* DList<T>::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}

#endif /* DLIST_H_ */
```

```
/*
 * SList.h
 *
 * Singly-linked list for type T
 *
 *  Created on: 17.02.2020
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_

#include <iostream>
#include <cassert>

#include "List.h"

template<typename T>
class SList: public List<T>
{
  public:
    SList() :
      List<T>(), pHead(nullptr)
    {

    }

    virtual ~SList();

    virtual void insertAt(int pos, const T& val);
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos);
    // deletes element at position pos (>0)

    virtual int search(const T& val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    virtual const T& getValue(int pos) const;
    // returns value at position pos

    virtual void setValue(int pos, const T& val);
    // sets value val at position pos

    virtual void print() const;
    // prints content of list to console

  private:
    struct Node
    {
        T value;
        Node* next;
    };
    Node* pHead; // ptr to head of list
    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

template<typename T>
SList<T>::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void SList<T>::insertAt(int pos, const T& val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  List<T>::setNumber(List<T>::getNumber() + 1);
}

template<typename T>
void SList<T>::deleteAt(int pos)
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  List<T>::setNumber(List<T>::getNumber() - 1);
}

template<typename T>
int SList<T>::search(const T& val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
const T& SList<T>::getValue(int pos) const
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  return nodePtr(pos)->value;
}

template<typename T>
void SList<T>::setValue(int pos, const T& val)
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  nodePtr(pos)->value = val;
}

template<typename T>
void SList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << List<T>::getNumber() << std::
      endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template <typename T>
typename SList<T>::Node* SList<T>::nodePtr(int pos) const
{
  assert(pos > 0 && pos <= List<T>::getNumber());
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}

#endif /* SLIST_H_ */
```

```cpp
/*
 * List.h
 *
 *  Created on: 15.02.2016
 *      Author: rbondere
 */

#ifndef LIST_H_
#define LIST_H_

template<typename Item>
class List
{
  public:
    List() :
        nr(0)
    {
    }
    virtual ~List()
    {
    }

    virtual void insertAt(int pos, const Item& val) = 0;
    // inserts element at (after) position pos (0: at head)

    virtual void deleteAt(int pos) = 0;
    // deletes element at position pos (>0)

    virtual int search(const Item& val) const = 0;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found

    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }

    int getNumber() const
    // returns number of elements
    {
      return nr;
    }

    virtual const Item& getValue(int pos) const = 0;
    // returns value at position pos

    virtual void setValue(int pos, const Item& val) = 0;
    // sets value val at position pos

    virtual void print() const = 0;
    // prints content of list to console

  protected:
    void setNumber(int n)
    // sets number of elements
    {
      nr = n;
    }

  private:
    int nr; // number of Elements
};

#endif /* LIST_H_ */
```

```cpp
//
      ============================================================================
// Name        : ListTest.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : List implementations
//
      ============================================================================


#include <iostream>
#include "SList.h"
#include "DList.h"
using namespace std;

int main()
{
  SList<double> s;
  cout << "Singly linked list" << endl;
  s.print();

  s.insertAt(0, 1.23);
  s.insertAt(0, -34.6);
  s.insertAt(0, 0.4);
  s.print();

  s.insertAt(3, 7.97);
  s.insertAt(1, 88.9);
  s.print();

  s.deleteAt(2);
  s.print();
  cout << "-----------------------------" << endl;

  DList<int> d;
  cout << "Doubly linked list" << endl;
  d.print();

  d.insertAt(0, 123);
  d.insertAt(0, -346);
  d.insertAt(0, 78);
  d.print();

  d.insertAt(3, 797);
  d.insertAt(1, 88);
  d.print();

  d.deleteAt(5);
  d.insertAt(4, 45);
  d.print();

  return 0;
}
```

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:

```
/*
 * MeasureList.h
 *
 *  Created on: 26.02.2015
 *      Author: rbondere
 *  Inheritance version
 */

#ifndef MEASURELIST_H_
#define MEASURELIST_H_

#include "SList.h"

class MeasureList: public SList<double>
{
  public:
    MeasureList(double tol = 20.0);
    void insertValue(double val);
    void setTolerance(double tol);
    double getMean(); // returns mean value
    void exclude(); // excludes values beyond tolerance band
  private:
    double tolerance; // in percent
};

#endif /* MEASURELIST_H_ */
```

```
/*
 * MeasureList.cpp
 *
 *  Created on: 26.02.2015
 *      Author: rbondere
 */

#include <cmath>
#include "MeasureList.h"

MeasureList::MeasureList(double tol) :
  tolerance(tol)
{
}

void MeasureList::setTolerance(double tol)
{
  tolerance = tol;
}

void MeasureList::exclude()
{
  double mean = getMean();
  // caution: mean may be negative
  double lower = mean - tolerance / 100.0 * fabs(mean);
  double higher = mean + tolerance / 100.0 * fabs(mean);
  for (int pos = 1; pos <= getNumber(); ++pos)
  {
    if (getValue(pos) < lower || getValue(pos) > higher)
    {
      deleteAt(pos);
      --pos;  // decrease position in this case
    }
  }
}

void MeasureList::insertValue(double val)
{
  insertAt(0, val); // inserts value at head
}

double MeasureList::getMean()
{
  double sum = 0.0;
  int nr = getNumber();
  for (int pos = 1; pos <= nr; ++pos)
  {
    sum += getValue(pos);
  }

  assert(nr);
  return sum / nr;
}
```

```
/*
 * SList.h
 *
 * Singly-linked list Template
 * The following operators must be defined for every type T
 * == << =
 *
 *  Created on: 20.02.2020
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_
#include <iostream>
#include <cassert>
template<typename T>
class SList
{
  public:
    SList() :
      pHead(nullptr), currPos(0), nr(0)
    {
    };
    ~SList();
    void insertAt(int pos,
                  T val);
    // inserts element at (after) position pos (0: at head)
    void deleteAt(int pos);
    // deletes element at position pos (>0)
    int search(T val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found
    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }
    int getNumber() const
    // returns number of elements
    {
      return nr;
    }
    int getNext() const;
    // returns position of next element (0: no next, i.e. at end)
    int getPosition() const
    // returns current position (0: list is empty)
    {
      return currPos;
    }
    void setPosition(int pos);
    // sets current position to pos
    T getValue(int pos) const;
    // returns value at position pos
    void setValue(int pos,
                  T val);
    // sets value val at position pos
    void print() const;
    // prints content of list to console
  private:
    struct Node
    {
        T value;
        Node* next;
    };
    Node* pHead; // ptr to head of list
    int currPos; // current position
    int nr; // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

// template implementation

template<typename T>
SList<T>::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void SList<T>::insertAt(int pos,
                        T val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
```

```
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  ++nr;
}

template<typename T>
void SList<T>::deleteAt(int pos)
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  --nr;
}

template<typename T>
int SList<T>::search(T val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
int SList<T>::getNext() const
{
  if (currPos >= nr)
    return 0;
  else
    return currPos + 1;
}

template<typename T>
void SList<T>::setPosition(int pos)
{
  assert(pos >= 0 && pos <= nr);
  currPos = pos;
}

template<typename T>
T SList<T>::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

template<typename T>
void SList<T>::setValue(int pos,
                        T val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

template<typename T>
void SList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << nr << std::endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template<typename T>
typename SList<T>::Node* SList<T>::nodePtr(int pos) const
// returns a pointer to the node given by position pos
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
#endif /* SLIST_H_ */
```

```
//
     ============================================================================
// Name        : MListMain.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Measurement list main program
//
     ============================================================================


#include <iostream>
#include "MeasureList.h"
using namespace std;

int main()
{
  MeasureList m;
  m.setTolerance(5);
  m.insertValue(9);
  m.insertValue(10);
  m.insertValue(11);
  m.print();
  cout << "Mean = " << m.getMean() << endl;
  m.exclude();
  m.print();
  m.exclude();
  m.print();
  m.insertValue(10.44);
  m.print();
  cout << "Mean = " << m.getMean() << endl;
  m.exclude();
  m.print();
  return 0;
}
```

Aggregation

```
/*
 * MeasureList.h
 *
 *  Created on: 24.02.2013
 *      Author: rbondere
 *  Aggregation version (no inheritance)
 */

#ifndef MEASURELIST_H_
#define MEASURELIST_H_

#include "SList.h"

class MeasureList
{
  public:
    MeasureList(double tol = 20.0);
    void insertValue(double val);
    void setTolerance(double tol);
    double getMean(); // returns mean value
    void exclude(); // excludes values beyond tolerance band
    void print() const; // prints content of list
  private:
    SList<double> list;
    double tolerance; // in percent
};

#endif /* MEASURELIST_H_ */
```

```cpp
/*
 * MeasureList.cpp
 *
 *  Created on: 24.02.2013
 *      Author: rbondere
 */

#include <cmath>
#include "MeasureList.h"

MeasureList::MeasureList(double tol) :
  tolerance(tol)
{
}

void MeasureList::setTolerance(double tol)
{
  tolerance = tol;
}

void MeasureList::exclude()
{
  double mean = getMean();
  // caution: mean may be negative
  double lower = mean - tolerance / 100.0 * fabs(mean);
  double higher = mean + tolerance / 100.0 * fabs(mean);
  for (int pos = 1; pos <= list.getNumber(); ++pos)
  {
    if (list.getValue(pos) < lower || list.getValue(pos) > higher)
    {
      list.deleteAt(pos);
      --pos;
    }
  }
}

void MeasureList::insertValue(double val)
{
  list.insertAt(0, val); // inserts value at head
}

void MeasureList::print() const
{
  list.print();
}

double MeasureList::getMean()
{
  double sum = 0;
  int nr = list.getNumber();
  for (int pos = 1; pos <= nr; ++pos)
  {
    sum += list.getValue(pos);
  }
  assert(nr);
  return sum / nr;
}
```

```cpp
/*
 * SList.h
 *
 * Singly-linked list Template
 * The following operators must be defined for every type T
 * == << =
 *
 *  Created on: 24.02.2020
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_
#include <iostream>
#include <cassert>
template<typename T>
class SList
{
  public:
    SList() :
      pHead(nullptr), currPos(0), nr(0)
    {
    };
    ~SList();
    void insertAt(int pos,
                  T val);
    // inserts element at (after) position pos (0: at head)
    void deleteAt(int pos);
    // deletes element at position pos (>0)
    int search(T val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found
    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }
    int getNumber() const
    // returns number of elements
    {
      return nr;
    }
    int getNext() const;
    // returns position of next element (0: no next, i.e. at end)
    int getPosition() const
    // returns current position (0: list is empty)
    {
      return currPos;
    }
    void setPosition(int pos);
    // sets current position to pos
    T getValue(int pos) const;
    // returns value at position pos
    void setValue(int pos,
                  T val);
    // sets value val at position pos
    void print() const;
    // prints content of list to console
  private:
    struct Node
    {
        T value;
        Node* next;
    };
    Node* pHead; // ptr to head of list
    int currPos; // current position
    int nr; // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

// template implementation

template<typename T>
SList<T>::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void SList<T>::insertAt(int pos,
                        T val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  ++nr;
}

template<typename T>
void SList<T>::deleteAt(int pos)
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  --nr;
}

template<typename T>
int SList<T>::search(T val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
```

```
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
int SList<T>::getNext() const
{
  if (currPos >= nr)
    return 0;
  else
    return currPos + 1;
}

template<typename T>
void SList<T>::setPosition(int pos)
{
  assert(pos >= 0 && pos <= nr);
  currPos = pos;
}

template<typename T>
T SList<T>::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

template<typename T>
void SList<T>::setValue(int pos,
                        T val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

template<typename T>
void SList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << nr << std::endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template<typename T>
typename SList<T>::Node* SList<T>::nodePtr(int pos) const
// returns a pointer to the node given by position pos
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
#endif /* SLIST_H_ */
```

```
//
      ============================================================================
// Name        : MListMain.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Measurement list main program
//
      ============================================================================


#include <iostream>
#include "MeasureList.h"
using namespace std;

int main()
{
  MeasureList m;
  m.setTolerance(5);
  m.insertValue(9);
  m.insertValue(10);
  m.insertValue(11);
  m.print();
  cout << "Mean = " << m.getMean() << endl;
  m.exclude();
  m.print();
  m.exclude();
  m.print();
  m.insertValue(10.44);
  m.exclude();
  m.print();
  return 0;
}
```

2.2 Aufgabe 2: Komplexitätsbetrachtungen

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

```
/ berechnet 'a^^b', Voraussetzung: b>=0
double potenz1(double a, int b)
{
    double c = 1;
    while (b > 0)
    {
        c = c * a;
        b = b - 1;
    }
    return c;
    }
        // berechnet 'a^^b', Voraussetzung: b>=0
        double potenz2(double a, int b)
        {
        if (b == 0)
        return 1;
        else
        return potenz2(a, b-1) * a;
    }
    // berechnet 'a^^b', Voraussetzung: b>=0
    double potenz3(double a, int b)
    {
        double c = 1;
        while (b > 0)
    {
        if (b % 2 == 1)
        c = c * a;
        a = a * a;
        b = b / 2;
    }
    reeturn c;
}
```

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 $ld(b)$ mal durchlaufen (ld = logarithmus dualis, Zweierlogarithmus). `potenz3()` ist demnach $O(logb)$.

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 verket-teten Liste arbeitet. Die `Stack`-Klasse soll als Template implementiert sein. Überlegen Sie sich auch hier als erstes, ob die Beziehung zur Listenklasse eine Vererbung oder eine Aggre-gation ist. Begründen Sie Ihre Wahl.

Als Methoden müssen Sie die bekannten Stackoperationen `push()`, `pop()`, `isEmpty()` und `peek()` realisie-ren. 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.

```
/*
 * SList.h
 *
 * Singly-linked list Template
 * The following operators must be defined for every type T
 * == << =
 *
 *  Created on: 20.02.20
 *      Author: rbondere
 */

#ifndef SLIST_H_
#define SLIST_H_
#include <cassert>
template<typename T>
class SList
{
  public:
    SList() :
      pHead(nullptr), currPos(0), nr(0)
    {
    };
    ~SList();
    void insertAt(int pos,
                  T val);
    // inserts element at (after) position pos (0: at head)
    void deleteAt(int pos);
    // deletes element at position pos (>0)
    int search(T val) const;
    // searches val in list and returns position of first match, starting
    // at head
    // returns 0 if value is not found
    bool isEmpty() const
    // returns true if list is empty, else false
    {
      return nr == 0;
    }
    int getNumber() const
    // returns number of elements
    {
      return nr;
    }
    int getNext() const;
    // returns position of next element (0: no next, i.e. at end)
    int getPosition() const
    // returns current position (0: list is empty)
    {
      return currPos;
    }
    void setPosition(int pos);
    // sets current position to pos
    T getValue(int pos) const;
    // returns value at position pos
    void setValue(int pos,
                  T val);
    // sets value val at position pos
    void print() const;
    // prints content of list to console
  private:
    struct Node
    {
        T value;
        Node* next;
    };
    Node* pHead; // ptr to head of list
    int currPos; // current position
    int nr; // number of Elements

    Node* nodePtr(int pos) const;
    // returns a pointer to the node given by position pos
};

// template implementation

template<typename T>
SList<T>::~SList()
{
  for (Node* p = pHead; p; p = pHead)
  {
    pHead = pHead->next;
    delete p;
  }
}

template<typename T>
void SList<T>::insertAt(int pos,
                        T val)
{
  assert(pos >= 0);
  Node* pEl = new Node;
  pEl->value = val;
  if (pos != 0)
  {
    Node* p = nodePtr(pos);
    assert(p != nullptr);
    pEl->next = p->next;
    p->next = pEl;
  }
  else // insert at head
  {
    pEl->next = pHead;
    pHead = pEl;
  }
  ++nr;
}

template<typename T>
void SList<T>::deleteAt(int pos)
```

```
{
  assert(pos > 0 && pos <= nr);
  Node* p = pHead; // cursor
  Node* pDel = p; // node to be deleted
  if (pos == 1) // first element
    pHead = pHead->next;
  else
  {
    for (int i = 1; i < pos - 1; ++i)
    {
      p = p->next;
    }
    pDel = p->next;
    p->next = pDel->next;
  }
  delete pDel;
  --nr;
}

template<typename T>
int SList<T>::search(T val) const
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p; ++i)
  {
    if (p->value == val)
      return i;
    p = p->next;
  }
  return 0; // not found
}

template<typename T>
int SList<T>::getNext() const
{
  if (currPos >= nr)
    return 0;
  else
    return currPos + 1;
}

template<typename T>
void SList<T>::setPosition(int pos)
{
  assert(pos >= 0 && pos <= nr);
  currPos = pos;
}

template<typename T>
T SList<T>::getValue(int pos) const
{
  assert(pos > 0 && pos <= nr);
  return nodePtr(pos)->value;
}

template<typename T>
void SList<T>::setValue(int pos,
                        T val)
{
  assert(pos > 0 && pos <= nr);
  nodePtr(pos)->value = val;
}

template<typename T>
void SList<T>::print() const
{
  std::cout << "--------------------" << std::endl;
  std::cout << "Number of elements: " << nr << std::endl;
  std::cout << "Content of list:" << std::endl;
  for (Node* p = pHead; p; p = p->next)
    std::cout << p->value << " ";
  std::cout << std::endl;
}

template<typename T>
typename SList<T>::Node* SList<T>::nodePtr(int pos) const
// returns a pointer to the node given by position pos
{
  Node* p = pHead; // p points to element at pos 1, if not empty
  for (int i = 1; p && i < pos; ++i)
  {
    p = p->next;
  }
  return p;
}
#endif /* SLIST_H_ */
```

```
/*
 * stack.h
 *
 *  Created on: 24.02.2017
 *      Author: rbondere
 *  Stack, der dynamisch erweitert wird
 */

#ifndef STACK_H_
#define STACK_H_

#include "SList.h"

template<typename ElemType>
class Stack
{
  public:
    Stack() : error(false)
    // Default-Konstruktor
    {
    }
    void push(const ElemType& e);
    // legt ein Element auf den Stack. Falls der Stack voll ist, wird er
    // dynamisch erweitert
    // wasError() gibt Auskunft, ob push() erfolgreich war

    ElemType pop();
    // nimmt ein Element vom Stack, falls der Stack nicht leer ist
    // wasError() gibt Auskunft, ob pop() erfolgreich war

    ElemType peek() const;
    // liest das oberste Element vom Stack, falls der Stack nicht leer
        ist
    // wasError() gibt Auskunft, ob peek() erfolgreich war

    bool isEmpty() const;
    // return: true: Stack ist leer
    //         false: sonst

    int getDepth() const;
    // return: Stacktiefe: Anzahl gueltige Elemente im Stack

    bool wasError() const
    // return: true: Operation war fehlerhaft
    //         false: sonst
    {
      return error;
    }
  private:
    SList<ElemType> list; // Dynamischer Speicher fuer Stack
    mutable bool error;   // true: Fehler passiert; false: sonst
    // mutable: auch const-Methoden koennen dieses Attribut setzen
};

// template implementation

template<typename ElemType>
void Stack<ElemType>::push(const ElemType& e)
{
  error = false;
  list.insertAt(0, e); // insert at head
}

template<typename ElemType>
ElemType Stack<ElemType>::pop()
{
  error = list.isEmpty();
  if (!error)
  {
    ElemType e = list.getValue(1); // pos == 1: element at head
    list.deleteAt(1);
    return e;
  }
  else
    return 0;
}

template<typename ElemType>
ElemType Stack<ElemType>::peek() const
{
  error = list.isEmpty();
  if (!error)
  {
    return list.getValue(1); // pos == 1: element at head
  }
  else
    return 0;
}

template<typename ElemType>
bool Stack<ElemType>::isEmpty() const
{
  return list.isEmpty();
}

template<typename ElemType>
int Stack<ElemType>::getDepth() const
{
  return list.getNumber();
}

#endif // STACK_H_
```

```cpp
/*
 * StackTest.cpp
 *
 *  Created on: 05.03.2020
 *      Author: rbondere
 */

#include <iostream>
#include <string>
#include "StackUI.h"
using namespace std;

int main()
{
  string theType = "";

  cout << "\n\nElementType (int, double, char): ";
  cin >> theType;
  if (theType == "int")
  {
    StackUI<int> sUI;
    sUI.dialog();
  }
  else if (theType == "double")
  {
    StackUI<double> sUI;
    sUI.dialog();
  }
  else if (theType == "char")
  {
    StackUI<char> sUI;
    sUI.dialog();
  }
  else
    cout << "Illegal input" << endl;

  cout << "Bye" << endl;

  return 0;
}
```

```cpp
/*
 * StackUI.h
 *
 * User Interface for Stack application
 * Created on: 24.02.2017
 *      Author: rbondere
 */

#ifndef STACKUI_H_
#define STACKUI_H_
#include "Stack.h"

template<typename ElemType>
class StackUI
{
  public:
    void dialog();
    // starts the user dialog
  private:
    Stack<ElemType> s;
};

// template implementation

// default Ctor of Stack is automatically called
#include <iostream>
using namespace std;

template<typename ElemType>
void StackUI<ElemType>::dialog()
{
  char ch = 0;
  ElemType e;
  do
  {
    cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty) ";
    cin >> ch;
    switch (ch)
    {
      case 'q':
      case 'Q': // quit
        break;
      case 'u':
      case 'U': // push
        cout << "\nElement to push: ";
        cin >> e;
        s.push(e);
        if (s.wasError())
          cout << "\nError: Stack full.";
        break;
      case 'o':
      case 'O': // pop
        e = s.pop();
        if (s.wasError())
          cout << "\nError: Stack is empty (nothing to pop).";
        else
          cout << "\nPopped element " << e;
        break;
      case 'k':
      case 'K': // peek
        e = s.peek();
        if (s.wasError())
          cout << "\nError: Stack is empty (nothing to peek).";
        else
          cout << "\nPeeked element " << e;
        break;
      case 'e':
      case 'E': // isEmpty
        if (s.isEmpty())
          cout << "\nStack is empty.";
        else
          cout << "\nStack contains " << s.getDepth() << " elements.";
        break;
      default:
        cout << "\nInvalid operation.";
        break;
    }
  } while (ch != 'Q' && ch != 'q');

}
#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.

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

Variante 2: Native Zeitmessung von Linux

```
#include <sys/resource.h>
#include <sys/types.h>
rusage tp;
double start; // Startzeit in Millisekunden
double end; // Endzeit
getrusage(RUSAGE_SELF, &tp);
start = static_cast<double>(tp.ru_utime.tv_sec) +
        static_cast<double>(tp.ru_utime.tv_usec)/1E6;
// do something
getrusage(RUSAGE_SELF, &tp);
end = static_cast<double>(tp.ru_utime.tv_sec) +
      static_cast<double>(tp.ru_utime.tv_usec)/1E6;
cout << "Dauer: " << end-start << " sec" << endl;
```

3.1.1 Lösung

Iterativ + Linux Zeitmessung

```
// Datei: fiboit.cpp
// berechnet die Fibonacci-Zahl iterativ
// R. Bonderer, 04.03.2013

#include <iostream>
#include <iomanip>
#include <sys/resource.h>
#include <sys/types.h>
using namespace std;

unsigned long fibonacci(unsigned int n);


int main(void)
{
  unsigned int zahl;
  unsigned long fibo;
  rusage tp;
  double start;   // Startzeit in Sekunden, Aufloesung in Mikrosekunden
  double end;     // Endzeit

  cout << "n eingeben: ";
  cin >> zahl;

  getrusage(RUSAGE_SELF, &tp);
  start = static_cast<double>(tp.ru_utime.tv_sec) +
          static_cast<double>(tp.ru_utime.tv_usec)/1E6;

  fibo = fibonacci(zahl);

  getrusage(RUSAGE_SELF, &tp);
  end = static_cast<double>(tp.ru_utime.tv_sec) +
        static_cast<double>(tp.ru_utime.tv_usec)/1E6;

  cout <<"Fibonacci(" << zahl << ")= " << fibo << endl;
  cout << fixed << setprecision(3) << "Dauer: " << end-start << " sec" <<
       endl;
  return 0;
}

unsigned long fibonacci(unsigned int n)
{
  unsigned long fibMin1 = 1;
  unsigned long fibMin2 = 1;
  unsigned long fib = 1;

  if (n == 1 || n == 2)
    return fib;

  for (unsigned int i = 3; i <= n; ++i)
  {
    fib = fibMin2 + fibMin1;
    fibMin2 = fibMin1;
    fibMin1 = fib;
  }
  return fib;
}
```

Rekursiv + `clock()` Zeitmessung

```
// Datei: fiborek.cpp
// berechnet die Fibonacci-Zahl rekursiv
// R. Bonderer, 03.03.2020

#include <iostream>
#include <ctime>

using namespace std;

// Funktionsprototypen
unsigned long fibonacci(unsigned int n);
// berechnet die Fibonacci-Zahl von n rekursiv
// return: Fibonacci-Zahl

int main()
{
  unsigned int zahl;
  unsigned long fibo;

  cout << "n eingeben: ";
  cin >> zahl;

  clock_t start = clock();
  fibo = fibonacci(zahl);
  clock_t end = clock();

  cout << "Fibonacci(" << zahl << ") = " << fibo << endl;
  cout << "Start: " << start << endl;
```

```
  cout << "End  : " << end << endl;
  cout << "Dauer: " << end-start << " Ticks" << endl;
  cout << "Dauer: " << static_cast<double>(end-start) / CLOCKS_PER_SEC
       << " sec" << endl;
  return 0;
}
```

```
unsigned long fibonacci(unsigned int n)
{
  if (n == 1 || n == 2)
    return 1;
  else
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Rekursiv + Linux Zeitmessung

```
// Datei: fiborek.cpp
// berechnet die Fibonacci-Zahl rekursiv
// misst Zeit mit Linux CPU Zeitfunktionen
// R. Bonderer, 03.03.2020

#include <iostream>
#include <iomanip>
#include <sys/resource.h>
#include <sys/types.h>

using namespace std;

// Funktionsprototypen
unsigned long fibonacci(unsigned int n);
// berechnet die Fibonacci-Zahl von n rekursiv
// return: Fibonacci-Zahl

int main()
{
  unsigned int zahl;
  unsigned long fibo;
  rusage tp;
  double start;   // Startzeit in Millisekunden
  double end;     // Endzeit

  cout << "n eingeben: ";
  cin >> zahl;

  getrusage(RUSAGE_SELF, &tp);
  start = static_cast<double>(tp.ru_utime.tv_sec) +
          static_cast<double>(tp.ru_utime.tv_usec)/1E6;

  fibo = fibonacci(zahl);

  getrusage(RUSAGE_SELF, &tp);
  end = static_cast<double>(tp.ru_utime.tv_sec) +
        static_cast<double>(tp.ru_utime.tv_usec)/1E6;

  cout << "Fibonacci(" << zahl << ") = " << fibo << endl;
  cout << fixed << setprecision(3) << "Dauer: " << end-start << " sec" <<
       endl;
  return 0;
}

unsigned long fibonacci(unsigned int n)
{
  if (n == 1 || n == 2)
    return 1;
  else
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

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:

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

3.2.1 Lösung

```
/*
 * StopWatch.h
 *
 *  Created on: 04.03.2013
 *      Author: rbondere
 */

#ifndef STOPWATCH_H_
#define STOPWATCH_H_

class StopWatch
{
  public:
    StopWatch();
    double elapsed() const;  // returns elapsed time since start in
          seconds
  private:
    double startTime;  // unit: seconds
};

#endif /* STOPWATCH_H_ */
```

```
/*
 * StopWatch.cpp
 *
 *  Created on: 04.03.2013
 *      Author: rbondere
 */

#include "StopWatch.h"
#include <sys/resource.h>
#include <sys/types.h>

StopWatch::StopWatch()
{
  rusage tp;
  getrusage(RUSAGE_SELF, &tp);
  startTime = static_cast<double>(tp.ru_utime.tv_sec) +
              static_cast<double>(tp.ru_utime.tv_usec)/1E6;
}

double StopWatch::elapsed() const
{
  rusage tp;
  getrusage(RUSAGE_SELF, &tp);
  return static_cast<double>(tp.ru_utime.tv_sec) +
         static_cast<double>(tp.ru_utime.tv_usec)/1E6 - startTime;
}
```

```
//
      ============================================================================
// Name        : StopWatchTest.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Test program for Stopwatch (Linux style)
//
      ============================================================================


#include <iostream>
#include "StopWatch.h"
```

```
using namespace std;
      ============================================================================
int main()
{
  StopWatch w;
  for (int j=0; j<10; ++j)
  {
    for(long i=0; i<100000000; ++i)
    {}
    cout << w.elapsed() << " sec." << endl;
  }

  return 0;
}
```

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)

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

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

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 lasst sich nicht geschlossen lösen.
solver: $\underline{\underline{n = 76127253}}$

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)

$k_5 \cdot 10^{10^6} = 1 \Rightarrow k_5 = \frac{1}{\underline{10^{10^6}}}$
$k_5 \cdot 10^n = 100$
$\Rightarrow 10^n = 100 \cdot 10^{10^6} = 10^{10^6+2}$
$\Rightarrow \underline{n} = 10^6 + 2 = \underline{1000002}$
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 Lis-te 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.

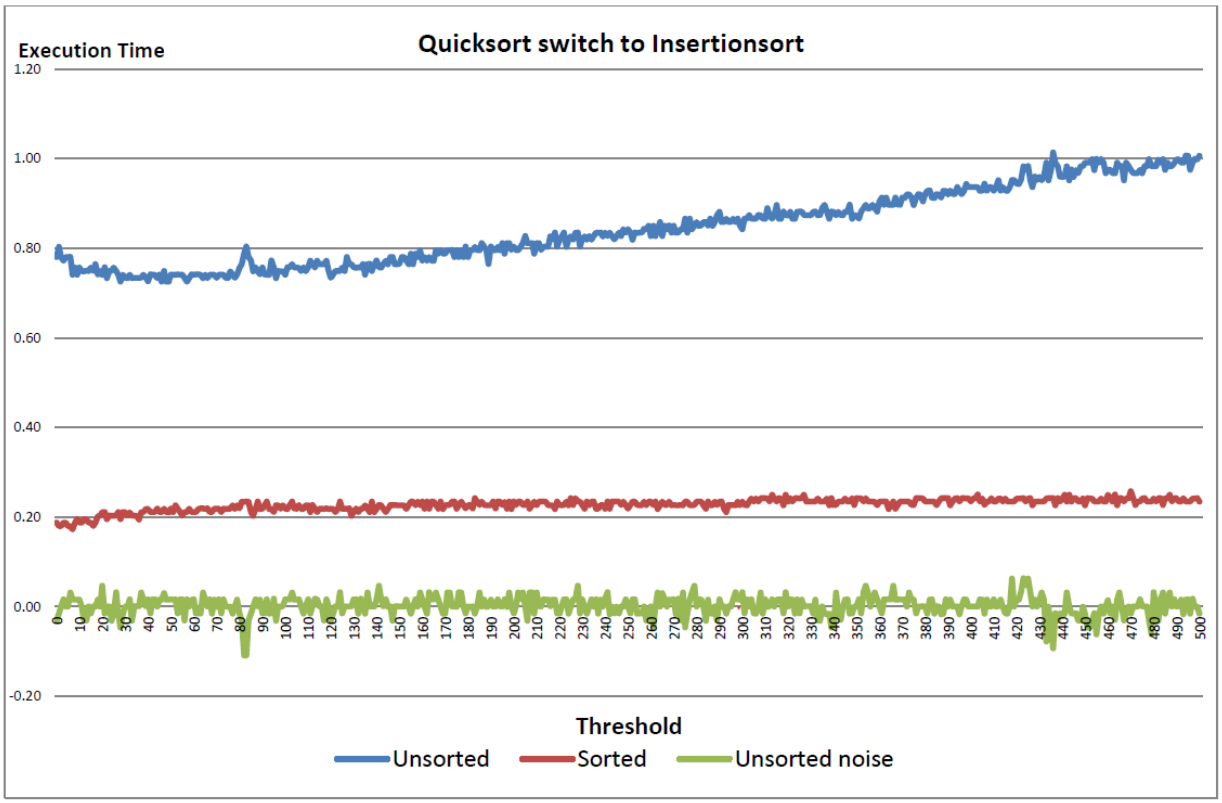


Abbildung 1: Laufzeitmessungen beim implementierten Algorithmus

```
/*
 * List.h
 *
 *  Created on: 01.03.2017
 *      Author: rbondere
 */

#ifndef LIST_H_
#define LIST_H_

class List
{
  public:
    List(int theSize = 100, int theThreshold = 1, int rSeed = 47);
    virtual ~List();
    void quickSort(int leftBound,
                   int rightBound);
    void insertionSort(int* d,          // start of list
                       int listLength);
    void randomFill(); // fill list with random values
  private:
    int size;  // size of list
    int threshold;
    int* data; // dynamic array
    int randomSeed;
    void swap(int& a,
              int& b);
};

#endif /* LIST_H_ */
```

```
/*
 * List.cpp
 *
 *  Created on: 01.03.2017
 *      Author: rbondere
 */

#include <cstdlib>
#include <ctime>
#include "List.h"

List::List(int theSize, int theThreshold, int rSeed) :
  size(theSize), threshold(theThreshold), data(new int[size]), randomSeed
      (rSeed)
{
}

List::~List()
{
  delete[] data;
}

```

```
void List::quickSort(int leftBound,
                     int rightBound)
{
  if (rightBound-leftBound < threshold)
    insertionSort(&data[leftBound], rightBound-leftBound+1);

  int left = leftBound; // index of left boundary
  int right = rightBound; // index of right boundary
  int pivot = data[(left + right) / 2]; // select middle element in array

  do
  {
    while (data[left] < pivot)
      ++left;
    while (data[right] > pivot)
      --right;

    if (left <= right)
    {
      if (left < right)
        swap(data[left], data[right]);
      ++left;
      --right;
    }
  } while (left <= right);

  if (leftBound < right)
    quickSort(leftBound, right);
  if (rightBound > left)
    quickSort(left, rightBound);

}
```

```
inline void List::insertionSort(int* d,
                                int listLength)
{
  int j;
  int buf; // Buffer

  for (int i = 1; i < listLength; i++)
  {
    buf = d[i];
    j = i;
    while (j > 0 && buf < d[j-1])
    {
      d[j] = d[j-1];
      j = j - 1;
    }
    d[j] = buf;
  }
}

void List::randomFill()
{
  srand(randomSeed);
  for (int i = 0; i < size; ++i)
    data[i] = rand();
}

void List::swap(int& a,
                int& b)
{
  int tmp = a;
  a = b;
  b = tmp;
}
```

```
//
      ============================================================================
// Name        : QuickSortFast.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Fast Quicksort
//
      ============================================================================


#include <iostream>
#include "StopWatch.h"
#include "List.h"
using namespace std;

int main()
{
  int nr = 5000000;
  int threshold;
  /*
   cout << "Anzahl Listenelemente: ";
   cin >> nr;
   */
  cout << "Schwelle fuer Umschaltung nach Insertionsort: ";
  cin >> threshold;

  List myList(nr, threshold);
  myList.randomFill();
```

```

  StopWatch t;
  myList.quickSort(0, nr - 1);
  cout << "Unsortierte Liste sortiert in " << t.elapsed() << " sec" <<
       endl;

  StopWatch t1;
  myList.quickSort(0, nr - 1);
  cout << "Sortierte Liste sortiert in   " << t1.elapsed() << " sec" <<
       endl;

//  for (threshold = 0; threshold <= 500; threshold++)
//  {
//    List myList(nr, threshold);
//    myList.randomFill();
//
//    StopWatch t;
//    myList.quickSort(0, nr - 1);
//    cout << threshold <<" " << t.elapsed();
//
//    StopWatch t1;
//    myList.quickSort(0, nr - 1); // already sorted
//    cout << " " << t1.elapsed() << endl;
//  }


  cout << "Fertig" << endl;
  return 0;
}
```

```
/*
 * StopWatch.h
 *
 *  Created on: 04.03.2013
 *      Author: rbondere
 */

#ifndef STOPWATCH_H_
#define STOPWATCH_H_
```

```
class StopWatch
{
  public:
    StopWatch();
    double elapsed() const;  // returns elapsed time since start
  private:
    double startTime;  // unit: seconds
};

#endif /* STOPWATCH_H_ */
```

```
/*
 * StopWatch.cpp
 *
 *  Created on: 04.03.2013
 *      Author: rbondere
 */

#include "StopWatch.h"
#include <sys/resource.h>
```

```
#include <sys/types.h>

StopWatch::StopWatch()
{
  rusage tp;
  getrusage(RUSAGE_SELF, &tp);
  startTime = static_cast<double>(tp.ru_utime.tv_sec) +
              static_cast<double>(tp.ru_utime.tv_usec)/1E6;
}
```

```
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 - startTime;
26  }
```

4 Lab 4 Zufallszahlengeneratoren

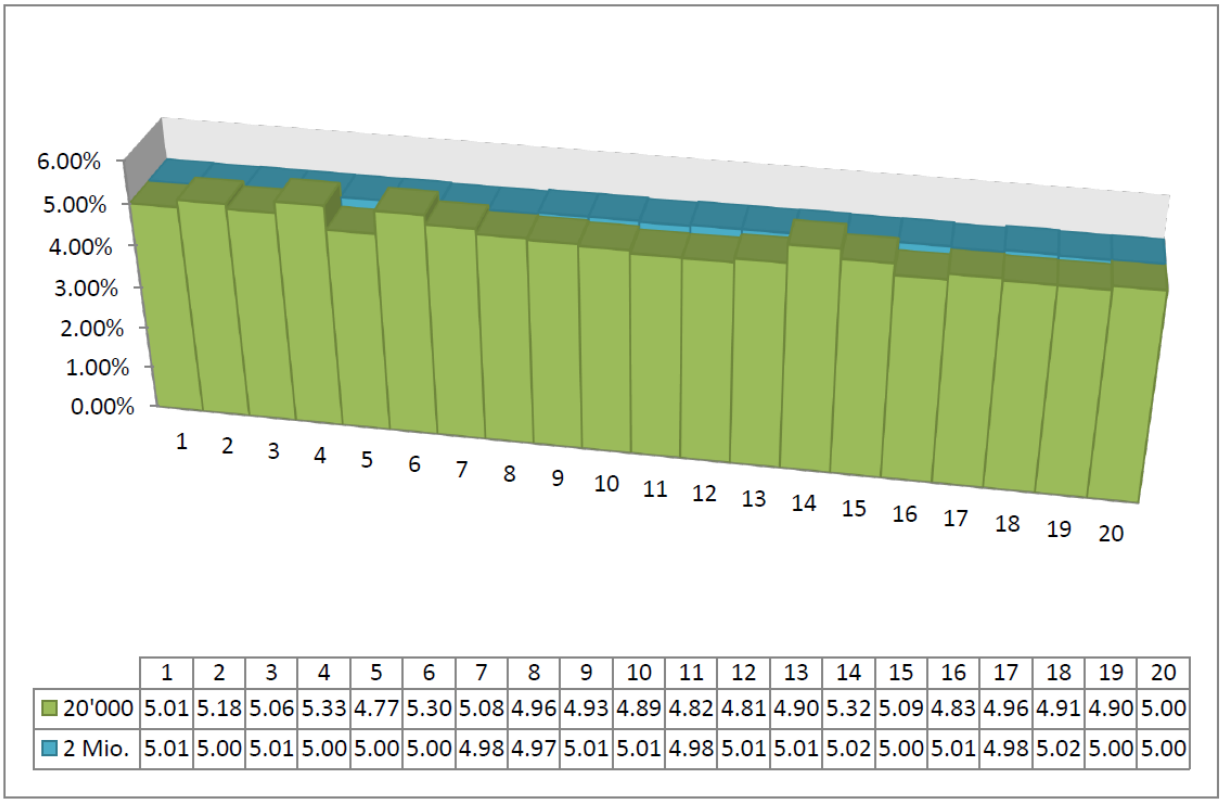
4.1 Aufgabe 1: Verteilung der Zufallswerte durch `rand()`

`rand()` liefert gleichverteilte Werte zwischen 0 und inklusive `RAND_MAX`, d.h im Intervall `[0, 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 Resul-tat 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 Be-reich ist wichtig, dass alle Werte, insbesondere auch der grösste und der kleinste, gleichverteilt sind. Die ein-fachste 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.

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



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
20'000	5.01	5.18	5.06	5.33	4.77	5.30	5.08	4.96	4.93	4.89	4.82	4.81	4.90	5.32	5.09	4.83	4.96	4.91	4.90	5.00
2 Mio.	5.01	5.00	5.01	5.00	5.00	5.00	4.98	4.97	5.01	5.01	4.98	5.01	5.01	5.02	5.00	5.01	4.98	5.02	5.00	5.00

dargestellt.

Abbildung 1: Verteilung der Zufallszahlen

```cpp
//
//============================================================================
// Name        : RandDist.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Uniform distribution
//
//============================================================================


#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

enum
{
  minValue = 1, // minimal value for random numbers
  maxValue = 20, // maximal value for random numbers
  iterations = 2000000 // number of iterations
};

int main()
{
  int histogram[maxValue - minValue + 1] = {0};
  int r;
  srand(time(0));
  for (int i = 0; i < iterations; ++i)
  {
    // r = rand() % (maxValue - minValue + 1) + minValue;
    // Modulo-Variante beruecksichtigt nur LSBs. Kann zu schlechten (kaum
    // zufaelligen) Folgen fuehren

    r = (int)(static_cast<double> (rand()) / (RAND_MAX + 1.0) * (maxValue
        - minValue + 1)) + minValue;
    // '+ 1' ist notwendig, weil nach int konvertiert wird.
    // Vor Konversion gibt es dadurch bei [1,20] Zahlen im Bereich
    // [0.0,20.0[
    // Nach Konversion werden nur die ganzzahligen Anteile verwendet
    // Das hinterste "+ minValue" muss ausserhalb des Typecasts sein,
    // andernfalls ergibt es ein Ueberhoehung bei 0, falls minValue < 0
    // cout << setw(2) << r << " | ";
    ++histogram[r - minValue];
  }
  for (int i = 0; i < maxValue - minValue + 1; i++)
    cout << histogram[i] << " ";
  cout << endl;

  cout << "Soll: je " << iterations / (maxValue - minValue + 1) << endl;
  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 Auf-gabe sollen Sie mit Hilfe des Computers einen graphischen Gütetest entwickeln und damit verschiedene Zu-fallszahlen-Generatoren testen. Verwenden Sie dazu Qt. Sie finden ein Qt Creator Projekt als Vorgabe.

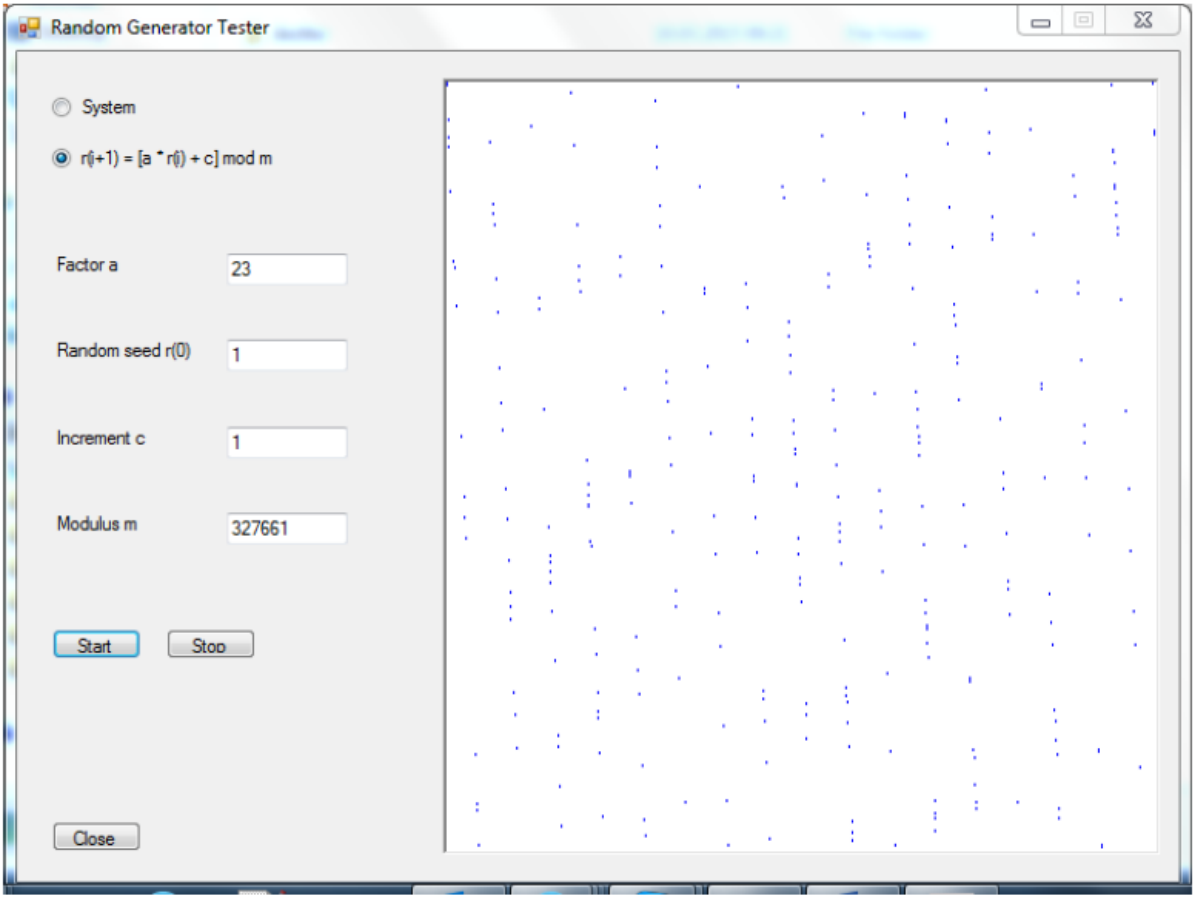


Abbildung 1: Beispielapplikation des Zufallszahlengenerator-Testers

1. Im graphischen Fenster soll ein Quadrat gezeichnet werden. Mittels zweier aufeinanderfolgender Zufalls-zahlen, die geeignet skaliert werden, wird ein Koordinatenpaar in diesem Quadrat festgelegt (1. Zufalls-zahl: x-Wert, 2. Zufallszahl: y-Wert) und an der entsprechenden Stelle ein Punkt gezeichnet. Im näch-sten 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 Anordneratoren 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

```cpp
#include "randomtester.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    RandomTester w;
    w.show();

    return a.exec();
}
```

```cpp
#ifndef RANDOMTESTER_H
#define RANDOMTESTER_H

#include <QMainWindow>
#include <QTimer>

#include <randomviewer.h>

namespace Ui {
class RandomTester;
}

class RandomTester : public QMainWindow
{
    Q_OBJECT

public:
    /**
     * @brief RandomTester Ctor
     * @param parent Parent
     */
    explicit RandomTester(QWidget *parent = 0);

    /**
      * @brief ~RandomTester Dtor
      */
    ~RandomTester();

public slots:
    /**
     * @brief onTimeout Draws a new random point
     */
    void onTimeout();

    /**
     * @brief onStartStopClicked Starts / stops the timer
     */
    void onStartStopClicked();

    /**
     * @brief onRadioClicked Select if the system random generator or
     * the equation r_i+1 = (a * r_i + c) mod m is used to generate the
          random values.
     */
    void onRadioClicked();

private:
    bool validateParameters();
    void invalidateParameters();

    enum RandomSource
    {
        srcSystem,
        srcModulo
    };

    Ui::RandomTester *ui;
    RandomViewer* viewer; /// RandomViewer to draw the random points on
         the pixmap
    QTimer* timer;

    RandomSource randomSrc;

    int xLim; /// x limit of the RandomViewer pixmap
    int yLim; /// y limit of the RandomViewer pixmap

    int x;
    int y;

    bool first;

    /// r = (a * r + c) mod m
    long factor; /// a
    long r; /// r
    long increment; /// c
    long modulus; /// m
};

#endif // RANDOMTESTER_H
```

```cpp
#include <QTime>

#include "randomtester.h"
#include "ui_randomtester.h"

RandomTester::RandomTester(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::RandomTester),
    viewer(new RandomViewer),
    randomSrc(srcModulo),
    first(true)
{
    ui->setupUi(this);

    switch (randomSrc)
    {
    case srcSystem:
        ui->radioSystem->setChecked(true);
        ui->radioModulo->setChecked(false);
        break;
    default:
    case srcModulo:
        ui->radioSystem->setChecked(false);
        ui->radioModulo->setChecked(true);
        break;
    }
    onRadioClicked();   // invoke function to update group box enabling

    timer = new QTimer();
    timer->setInterval(10);
    connect(timer, SIGNAL(timeout()), this, SLOT(onTimeout()));

    // Initalize graphics window
    ui->centralWidget->layout()->addWidget(viewer);

    xLim = viewer->size().width();
    yLim = viewer->size().height();

    // Initialize system random
    qsrand(QDateTime::currentDateTime().toMSecsSinceEpoch());

    connect(ui->btnStartStop, SIGNAL(clicked()), this, SLOT(
        onStartStopClicked()));
    connect(ui->radioSystem, SIGNAL(clicked()), this, SLOT(onRadioClicked
        ()));
    connect(ui->radioModulo, SIGNAL(clicked()), this, SLOT(onRadioClicked
        ()));
}

RandomTester::~RandomTester()
{
    if (timer->isActive())
        timer->stop();

    delete ui;
    delete timer;
    delete viewer;
}

void RandomTester::onTimeout()
{
    x = y;
    switch (randomSrc)
    {
    case srcSystem:
        y = (1.0 * qrand() / RAND_MAX) * yLim;
        break;
    default:
    case srcModulo:
        r = (factor * r + increment) % modulus;
        y = static_cast<int>(r) * yLim / modulus;
        break;
    }
    if (!first)
        viewer->addPoint(QPoint(x, y));
    first = false;
}

void RandomTester::onStartStopClicked()
{
    if (timer->isActive())
    {
        timer->stop();
        ui->btnStartStop->setText("Start");
        if (ui->radioModulo->isChecked())
            ui->gbParameters->setEnabled(true);
        ui->gbSource->setEnabled(true);
    }
    else
    {
        if (validateParameters())
        {
            viewer->clear();
            first = true;
            timer->start();
            ui->btnStartStop->setText("Stop");
            ui->gbSource->setEnabled(false);
            ui->gbParameters->setEnabled(false);
        }
    }
}

void RandomTester::onRadioClicked()
{
    if (ui->radioSystem->isChecked())
    {
        ui->gbParameters->setEnabled(false);
        randomSrc = srcSystem;
    }
    else if (ui->radioModulo->isChecked())
    {
        ui->gbParameters->setEnabled(true);
        randomSrc = srcModulo;
    }
}

bool RandomTester::validateParameters()
{
    bool res;
    bool resAll = true;

    invalidateParameters();

    factor = ui->leFactor->text().toInt(&res);
    if (!res)
    {
        ui->leFactor->setStyleSheet("background-color: red;");
        resAll = false;
    }

    r = ui->leSeed->text().toInt(&res);
    if (!res)
    {
        ui->leSeed->setStyleSheet("background-color: red;");
        resAll = false;
    }

    increment = ui->leIncrement->text().toInt(&res);
    if (!res)
    {
        ui->leIncrement->setStyleSheet("background-color: red;");
        resAll = false;
    }

    modulus = ui->leModulus->text().toInt(&res);
    if (!res)
    {
        ui->leModulus->setStyleSheet("background-color: red;");
        resAll = false;
    }

    return resAll;
}

void RandomTester::invalidateParameters()
{
    ui->leFactor->setStyleSheet("");
    ui->leSeed->setStyleSheet("");
    ui->leIncrement->setStyleSheet("");
    ui->leModulus->setStyleSheet("");
}
```

```cpp
#ifndef RANDOMVIEWER_H
#define RANDOMVIEWER_H

#include <QFrame>
#include <QPixmap>
#include <QSize>
#include <QPen>

class RandomViewer : public QFrame
{
    Q_OBJECT
public:
    /**
     * @brief RandomViewer Ctor
     * @param s Size of the pixmap
     * @param parent Parent
     */
    explicit RandomViewer(QSize s = QSize(600, 600), QWidget *parent = 0)
        ;

    /**
     * @brief addPoint Draws the point in the pixmap
     * @param p Point to draw
     */
    void addPoint(QPoint p);

    /**
     * @brief clear Clears the pixmap
     */
    void clear();

signals:
```

```
32
33  public slots:
34
35  protected:
36      void paintEvent(QPaintEvent *);
37
38  private:
39      QPen pen;
40      QPixmap pixmap;
41  };
42
43  #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 Mün-ze implementiert. Studieren Sie den Code und verifizieren Sie, ob die Verteilung den Erwartungen ent-spricht.

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 Funk-tion `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.

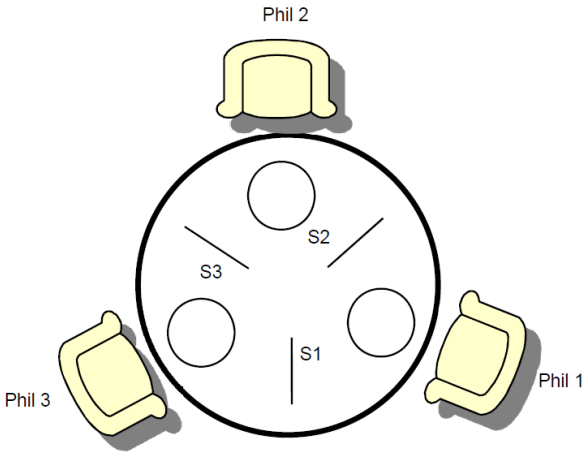
Der Code ist unter ./Loesung/Coin/fairCoin.cpp zu finden.

Der Output kann nun so aussehen:
0: 5000155 (50.0016 %)
1: 4999845 (49.9984 %)

```
//
     ============================================================================
// Name        : biasedCoin.cpp
// Author      : Reto Bonderer
// Version     : 20190528
// Copyright   : (c) HSR R. Bonderer
// Description : Get a uniform distribution from a biased coin
//
     ============================================================================


#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

enum
{
  iterations = 10000000 // number of iterations
};

// returns a biased coin value (0, 1)
unsigned int biasedCoin();

// returns a fair coin value (0, 1) using biasedCoin()
unsigned int fairCoin();

int main()
{
  unsigned int histogram[2] = {0};
  srand(time(0));

  for (unsigned int i = 0; i < iterations; ++i)
  {
    ++histogram[fairCoin()];
  }
  for (int i = 0; i < 2; ++i)
    cout << i << ": " << setw(10) << histogram[i] << " ("
         << histogram[i]*100.0/iterations << " %)" << endl;

  return 0;
}

     ============================================================================
unsigned int biasedCoin()
{
  enum
  {
    bias = 60   // p(0) = bias in percent, p(1) = 1-p(0)
  };

  if (rand() < RAND_MAX/100*bias)
    return 0;
  else
    return 1;
}

unsigned int fairCoin()
{
  while (1)
  {
    unsigned int first = biasedCoin();
    unsigned int second = biasedCoin();
    if (first == 0 && second == 1)
      return 0;
    else if (first == 1 && second == 0)
      return 1;
  }
}
```

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

m 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.

```
{
  public:
      Philosopher(int pid, int thinkdelay, int eatDelay, Sticks& s);
      ~Philosopher();
      void live();        // the philosopher's life
      void join();        // wait for philosopher to leave
  private:
      enum {nMeals = 3};
      int id;             // this philosopher's id
      int tDelay;         // how long does this philosopher think?
      int eDelay;         // how long does this philosopher eat?
      int left;           // left fork number
      int right;          // right fork number
      Sticks& stick;      // sticks used by all philosophers
      pthread_attr_t attr;
      pthread_t tid;      // thread id
      void lifeThread(); // the (C++) - thread function
      static void* staticWrapper(void* p) // C Wrapper for pthread_create()
      {                                   // p must be the this pointer
          static_cast<Philosopher*>(p) -> lifeThread();
          return 0;
      }
};
```

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::lifeThr`
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::live()`, 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::lifeThre` 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 threadsafe ist. Einzelne Texte mögen auseinandergerissen (interleaved) sein.

```
/*
 * Sticks.h
 *
 * Dining Philosophers: Sticks class (Monitor)
 *
 *  Created on: 07.04.2016
 *      Author: Reto Bonderer
 */
#ifndef STICKS_H_
#define STICKS_H_

class Sticks
{
  public:
    Sticks(int nr = 5);
    ~Sticks();
    int getNr() const {return nSticks;}
    void put(int left, int right);  // lay down left and right stick
    void get(int left, int right);  // try to pick left and right
         stick
  private:
    int nSticks;
    volatile bool* stick;  // ptr to dynamic array
    pthread_mutex_t stickMutex;
    pthread_cond_t  stickCv;
};

#endif
```

```
/*
 * Sticks.cpp
 *
 * Dining Philosophers: Sticks class (Monitor)
 *
 *  Created on: 07.04.2016
 *      Author: Reto Bonderer
 */

#include <pthread.h>
#include "Sticks.h"

Sticks::Sticks(int nr)
  : nSticks(nr),
    stick(new bool[nSticks])
{
  for (int i = 0; i < nSticks; ++i)
    stick[i] = false;

  // Initialize mutex and condition variable objects
  pthread_mutex_init(&stickMutex, 0);
  pthread_cond_init (&stickCv, 0);
}

Sticks::~Sticks()
{
  delete[] stick;
  pthread_mutex_destroy(&stickMutex);
  pthread_cond_destroy(&stickCv);
}

void Sticks::put(int left, int right)
{
  pthread_mutex_lock(&stickMutex);        // start of CS
  stick[left] = false;
  stick[right] = false;
  pthread_cond_signal(&stickCv);
  pthread_mutex_unlock(&stickMutex);      // end of CS
}

void Sticks::get(int left, int right)
{
  pthread_mutex_lock(&stickMutex);        // start of CS
  while (stick[left] || stick[right])
  {
    pthread_cond_wait(&stickCv, &stickMutex);
  }
  stick[left] = true;
  stick[right] = true;
  pthread_mutex_unlock(&stickMutex);      // end of CS
}
```

```
/*
 * Philosopher.h
 *
 * Dining Philosophers: Philosopher class
 *
 *  Created on: 20.12.2016
 *      Author: Reto Bonderer
 */
#ifndef PHILOSOPHER_H_
#define PHILOSOPHER_H_
#include <pthread.h>
#include "Sticks.h"

class Philosopher
{
  public:
    Philosopher(int pid, int thinkdelay, int eatDelay, Sticks& s);
    ~Philosopher();
    void live();        // the philosopher's life
    void join();        // wait for philosopher to leave
  private:
    enum {nMeals = 3};
    int id;             // this philosopher's id
    int tDelay;         // how long does this philosopher think?
    int eDelay;         // how long does this philosopher eat?
    int left;           // left fork number
    int right;          // right fork number
    Sticks& stick;      // sticks used by all philosophers
    pthread_attr_t attr;
    pthread_t tid;      // thread id
    void lifeThread();  // the (C++) - thread function
    static void* staticWrapper(void* p) // C Wrapper for
         pthread_create()
         {                              // p must be the this
               pointer
            static_cast<Philosopher*>(p)->lifeThread();
           return 0;
         }
};

#endif
```

```cpp
/*
 * Philosopher.cpp
 *
 * Dining Philosophers: Philosopher class
 *
 *  Created on: 24.03.2020
 *      Author: Reto Bonderer
 */
#include <iostream>
#include <unistd.h>
#include "Philosopher.h"
#include "Sticks.h"
using namespace std;

Philosopher::Philosopher(int pid, int thinkDelay, int eatDelay, Sticks& s)
  : id(pid),
    tDelay(thinkDelay),
    eDelay(eatDelay),
    stick(s)
{
  left = id==1? stick.getNr() - 1: id-2; // follow the anti deadlock strategy
  right = id-1;
}

Philosopher::~Philosopher()
{
  pthread_attr_destroy(&attr);
}

void Philosopher::live()
{
  // For portability, explicitly create threads in a joinable state
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  pthread_create(&tid, &attr, staticWrapper, this); // call the static wrapper
}

void Philosopher::join()
{
  pthread_join(tid, 0);
}

void Philosopher::lifeThread()
{
  for (int i = 0 ; i < nMeals; ++i)
  {
    cout << "Philosopher " << id << " is thinking..." << endl;
    usleep(tDelay);
    cout << "Philosopher " << id << " likes to eat..." << endl;
    stick.get(left, right);
    cout << "Philosopher " << id << " is eating..." << endl;
    usleep(eDelay);
    cout << "Philosopher " << id << " puts sticks back..." << endl;
    stick.put(left, right);
  }
  cout << "Philosopher " << id << " leaves." << endl;
  pthread_exit(0);
}
```

```cpp
/*
 * PhiloTest.cpp
 *
 * Dining Philosophers: Philosopher test program
 *
 *
 *  Created on: 24.03.2020
 *      Author: Reto Bonderer
 */
#include <pthread.h>
#include <unistd.h>
#include <iostream>
#include "Philosopher.h"
#include "Sticks.h"
using namespace std;

enum {numPhilos = 5};

int main()
{
  Sticks s(numPhilos);  // shared resource
  Philosopher p[numPhilos] =
  {   // pid, thinkDelay, eatDelay, Sticks&
    Philosopher(1, 100000, 500000, s),
    Philosopher(2, 200000, 400000, s),
    Philosopher(3, 300000, 300000, s),
    Philosopher(4, 400000, 200000, s),
    Philosopher(5, 500000, 100000, s)
  };

  for (int i = 0 ; i < numPhilos; ++i)
  {
    p[i].live();
  }

  // wait for all philosophers to leave
  for (int i = 0 ; i < numPhilos; ++i)
  {
    p[i].join();
  }
  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.

```cpp
/*
 * Sticks.h
 *
 * Dining Philosophers: Sticks class (Monitor)
 *
 *  Created on: 07.04.2016
 *      Author: Reto Bonderer
 */
#ifndef STICKS_H_
#define STICKS_H_

class Sticks
{
  public:
    Sticks(int nr = 5);
    ~Sticks();
    int getNr() const {return nSticks;}
    void put(int left, int right);  // lay down left and right stick
    void get(int left, int right);  // try to pick left and right stick
  private:
    int nSticks;
    volatile bool* stick;    // ptr to dynamic array
    pthread_mutex_t stickMutex;
    pthread_cond_t  stickCv;
};

#endif
```

```cpp
/*
 * Sticks.cpp
 *
 * Dining Philosophers: Sticks class (Monitor) using RAII
 *
 *  Created on: 25.03.2020
 *      Author: Reto Bonderer
 */

#include <pthread.h>
```

```cpp
#include "Sticks.h"
#include "ResourceLock.h"

Sticks::Sticks(int nr)
  : nSticks(nr),
    stick(new bool[nSticks])
{
  for (int i = 0; i < nSticks; ++i)
    stick[i] = false;

  // Initialize mutex and condition variable objects
  pthread_mutex_init(&stickMutex, 0);
  pthread_cond_init (&stickCv, 0);
}

Sticks::~Sticks()
{
  delete[] stick;
  pthread_mutex_destroy(&stickMutex);
  pthread_cond_destroy(&stickCv);
}

void Sticks::put(int left, int right)
{
  {                                        // start of CS
                ResourceLock lock(stickMutex);
                stick[left] = false;
                stick[right] = false;
                pthread_cond_signal(&stickCv);
  }                                        // end of CS
}

void Sticks::get(int left, int right)
{
  {                                        // start of CS
                ResourceLock lock(stickMutex);
                while (stick[left] || stick[right])
                {
                  pthread_cond_wait(&stickCv, &stickMutex);
                }
                stick[left] = true;
                stick[right] = true;
  }                                        // end of CS
}
```

```cpp
/*
 * Philosopher.h
 *
 * Dining Philosophers: Philosopher class
 *
 *  Created on: 20.12.2016
 *      Author: Reto Bonderer
 */
#ifndef PHILOSOPHER_H_
#define PHILOSOPHER_H_
#include <pthread.h>
#include "Sticks.h"

class Philosopher
{
  public:
    Philosopher(int pid, int thinkdelay, int eatDelay, Sticks& s);
    ~Philosopher();
    void live();        // the philosopher's life
    void join();        // wait for philosopher to leave
  private:
    enum {nMeals = 3};
    int id;             // this philosopher's id
    int tDelay;         // how long does this philosopher think?
    int eDelay;         // how long does this philosopher eat?
    int left;           // left fork number
    int right;          // right fork number
    Sticks& stick;      // sticks used by all philosophers
    pthread_attr_t attr;
    pthread_t tid;      // thread id
    void lifeThread();  // the (C++) - thread function
    static void* staticWrapper(void* p) // C Wrapper for pthread_create()
          {                                  // p must be the this
                pointer
                  static_cast<Philosopher*>(p)->lifeThread();
                return 0;
          }
};

#endif
```

```cpp
/*
 * Philosopher.cpp
 *
 * Dining Philosophers: Philosopher class
 *
 *  Created on: 24.03.2020
 *      Author: Reto Bonderer
 */
#include <iostream>
#include <unistd.h>
#include "Philosopher.h"
#include "Sticks.h"
using namespace std;

Philosopher::Philosopher(int pid, int thinkDelay, int eatDelay, Sticks& s
    )
  : id(pid),
    tDelay(thinkDelay),
    eDelay(eatDelay),
    stick(s)
{
  left = id==1? stick.getNr() - 1: id-2; // follow the anti deadlock
      strategy
  right = id-1;
}

Philosopher::~Philosopher()
{
  pthread_attr_destroy(&attr);
}

void Philosopher::live()
{
  // For portability, explicitly create threads in a joinable state
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  pthread_create(&tid, &attr, staticWrapper, this); // call the static
      wrapper
}

void Philosopher::join()
{
  pthread_join(tid, 0);
}

void Philosopher::lifeThread()
{
  for (int i = 0 ; i < nMeals; ++i)
  {
    cout << "Philosopher " << id << " is thinking..." << endl;
    usleep(tDelay);
    cout << "Philosopher " << id << " likes to eat..." << endl;
    stick.get(left, right);
    cout << "Philosopher " << id << " is eating..." << endl;
    usleep(eDelay);
    cout << "Philosopher " << id << " puts sticks back..." << endl;
    stick.put(left, right);
  }
  cout << "Philosopher " << id << " leaves." << endl;
  pthread_exit(0);
}
```

```cpp
/*
 * PhiloTest.cpp
 *
 * Dining Philosophers: Philosopher test program
 *
 *
 *  Created on: 24.03.2020
 *      Author: Reto Bonderer
 */

#include <pthread.h>
#include <unistd.h>
#include <iostream>
#include "Philosopher.h"
#include "Sticks.h"
using namespace std;

enum {numPhilos = 5};
```

```cpp
int main()
{
  Sticks s(numPhilos);  // shared resource
  Philosopher p[numPhilos] =
  {   // pid, thinkDelay, eatDelay, Sticks&
    Philosopher(1, 100000, 500000, s),
    Philosopher(2, 200000, 400000, s),
    Philosopher(3, 300000, 300000, s),
    Philosopher(4, 400000, 200000, s),
    Philosopher(5, 500000, 100000, s)
  };

  for (int i = 0 ; i < numPhilos; ++i)
  {
    p[i].live();
  }

  // wait for all philosophers to leave
  for (int i = 0 ; i < numPhilos; ++i)
  {
    p[i].join();
  }
  return 0;
}
```

```cpp
/*
 * ResourceLock.h
 * RAII implementation of lock/unlock mutex
 *
 *  Created on: Apr 08, 2019
 *      Author: reto.bonderer
 */

#ifndef RESOURCELOCK_H_
#define RESOURCELOCK_H_
#include <pthread.h>
class ResourceLock
{
  public:
    ResourceLock(pthread_mutex_t& m) :
        mutex(m)
    {
      pthread_mutex_lock(&mutex);
    }
    ~ResourceLock()
    {
      pthread_mutex_unlock(&mutex);
    }
  private:
    pthread_mutex_t& mutex; // mutex of shared resource
};

#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      =   CRC
        1 0 0 0 0 0 1 1 1            =   XOR Polynomial
        -----------------
        0 1 0 1 1 1 0 1 1 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.

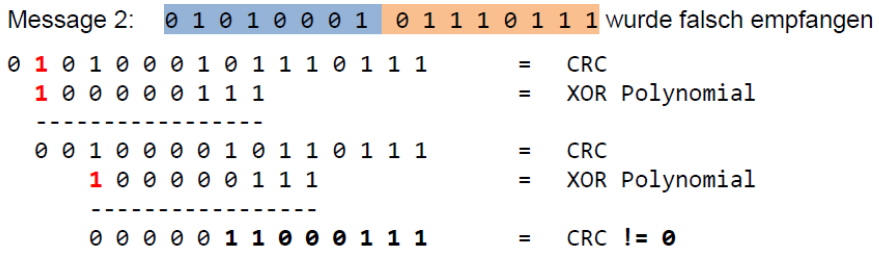
6.2.1 Lösung

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



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.

```
$ ./crcTest
CRC-8 CCITT Bit um Bit, einfach aber ineffizient

CRC-8 CCITT der einzelnen Datenbytes:
Byte 00: crc(0xd9) = 0x01
Byte 01: crc(0x51) = 0xb0
Byte 02: crc(0x61) = 0x20
CRC-8 CCITT von allen Datenbytes:
crc(0xd95161) = 0x2c
```

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

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

```
/*
 * Crc.h
 *
 * CRC 8 bit simple implementation
 *
 *  Created on: 25.03.2019
 *      Author: Reto Bonderer
 */
#ifndef CRC_H_
#define CRC_H_

#include <stdint.h>

typedef uint8_t CrcType;

class Crc
{
  public:
    Crc(CrcType thePoly = 0x07);
    CrcType getCrc(const uint8_t message[], unsigned int nBytes) const;
  private:
    CrcType poly;
};

 #endif // CRC_H_
```

```cpp
/*
 * Crc.cpp
 *
 *  Created on: 25.03.2019
 *      Author: Reto Bonderer
 */

#include "Crc.h"

enum {crcWidth = 8 * sizeof(CrcType),
      crcTopBit = 1 << (crcWidth - 1)};

Crc::Crc(CrcType thePoly) :
  poly(thePoly)
{
}

CrcType Crc::getCrc(const uint8_t message[], unsigned int nBytes) const
{
  CrcType remainder = 0;

  // modulo-2 division; byte by byte
  for (unsigned int byte = 0; byte < nBytes; ++byte)
  {
    // XOR the next byte to the remainder
    remainder ^= (message[byte] << (crcWidth - 8));

    // modulo-2 division; bit by bit
    for (int bit = 8; bit > 0; --bit)
    {
      if (remainder & crcTopBit)
      {
        remainder = (remainder << 1) ^ poly;
      }
      else
      {
        remainder = (remainder << 1);
      }
    }
  }
  return remainder;
}
```

```cpp
/*
 * crcTest.h
 *
 *  Created on: 25.03.2019
 *      Author: Reto Bonderer
 */

#include <iostream>
#include <fstream>
#include <iomanip>
#include "Crc.h"

using namespace std;

int main(int argc, char* argv[])
{
  // Predefined data will be used if no file name has been passed.
  const uint8_t data[] = {0xd9, 0x51, 0x61};

  Crc crc(0x07);  // use CRC-8 CCITT
  CrcType crcValue;

  cout << "CRC-8 CCITT Bit um Bit, einfach aber ineffizient" << endl <<
      endl;

  if (argc > 1)
  {
    // compute CRC of the passed file (argv[1])
    ifstream f;
    uint8_t* buf;
    unsigned int len;

    try
    {
      f.open(argv[1], ios::in | ios::binary);

      // determine file size
      f.seekg(0, ios::end);
      len = f.tellg();
      f.seekg(0, ios::beg);

      // read file content into byte buffer
      buf = new uint8_t[len];
      f.read((char*)buf, len);
      f.close();

      cout << "Datei:              " << argv[1] << endl;
      cout << "Anzahl Datenbytes: " << len << endl;

      // calculate CRC of the whole byte buffer
      crcValue = crc.getCrc(buf, len);

      cout << "CRC-8 CCITT:        " << setfill('0') << hex << showbase <<
          internal
          << setw(4) << (int)crcValue << endl;

      // free allocated memory
      delete[] buf;
    }
    catch (const exception& ex)
    {
      cerr << "Unable to read file '" << argv[1] << "'!" << endl;
    }
  }
  else
  {
    cout << "CRC-8 CCITT der einzelnen Datenbytes:" << endl;

    for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]); ++i)
    {
      // compute CRC of the individual data bytes
      crcValue = crc.getCrc(&data[i], 1);

      cout << "Byte " << dec << setfill('0') << setw(2) << i
          << ": crc(" << showbase << hex << internal << setw(4) << (int)
              data[i]
          << ") = " << setw(4) << (int)crcValue << endl;
    }

    // compute CRC of all data bytes
    crcValue = crc.getCrc(data, sizeof(data)/sizeof(data[0]));

    cout << "CRC-8 CCITT von allen Datenbytes:" << endl;
    cout << "crc(" << setfill('0') << showbase << hex << internal << setw
        (4);
    for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]); ++i)
    {
      cout << (int)data[i] << noshowbase << setw(2);
    }
    cout << ") = " << showbase << setw(4) << (int)crcValue << endl <<
        endl;
  }
  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/=:

```
#include <fstream>
 ...
f.open(argv[1], ios::in | ios::binary);
// determine file size
f.seekg(0, ios::end);
len = f.tellg();
f.seekg(0, ios::beg);
// Allocate byte buffer
buf = new uint8_t[len];
// read file content into byte buffer
f.read((char*)buf, len);
f.close();
// ...
delete[] buf; // free allocated memory
```

6.4.1 Lösung

```
/*
 * Crc.h
 *
 * CRC 8 bit lookup table implementation
 *
 *  Created on: 25.03.2019
 *      Author: Reto Bonderer
 */
#ifndef CRC_H_
#define CRC_H_

#include <stdint.h>

typedef uint8_t CrcType;

class Crc
{
  public:
    Crc(CrcType thePoly = 0x07);
    CrcType getCrc(const uint8_t message[], unsigned int nBytes) const;
  private:
    CrcType poly;
    CrcType crcTable[256];
    void computeTable();
};

#endif // CRC_H_
```

```
/*
 * crc.cpp
 *
 *  Created on: 31.03.2016
 *      Author: Reto Bonderer
 */

#include <iostream>
#include <iomanip>
#include "Crc.h"
using namespace std;

enum {crcWidth = 8 * sizeof(CrcType),
      crcTopBit = 1 << (crcWidth - 1)};

Crc::Crc(CrcType thePoly) :
  poly(thePoly)
{
  computeTable();
}

void Crc::computeTable()
{
  CrcType remainder = 0;

  for (unsigned int i = 0; i < 256; ++i)
  {
    remainder = i << (crcWidth - 8);
    for (unsigned int bit = 8; bit > 0; --bit)
    {
      if (remainder & crcTopBit)
      {
        remainder = (remainder << 1) ^ poly;
      }
      else
      {
        remainder <<= 1;
      }
    }
    crcTable[i] = remainder;
  }
}

CrcType Crc::getCrc(const uint8_t message[], unsigned int nBytes) const
{
  CrcType remainder = 0;
  uint8_t data;

  for (unsigned int byte = 0; byte < nBytes; ++byte)
  {
    data = message[byte] ^ (remainder >> (crcWidth - 8));
    remainder = crcTable[data] ^ (remainder << 8);
  }

  return remainder;
}
```

```
/*
 * crcTest.h
 *
 *  Created on: 31.03.2016
 *      Author: Reto Bonderer
 */

#include <iostream>
#include <fstream>
#include <iomanip>
#include "Crc.h"

using namespace std;

int main(int argc, char* argv[])
{
  // Predefined data will be used if no file name has been passed.
  const uint8_t data[] = {0xd9, 0x51, 0x61};

  Crc crc(0x07);  // use CRC-8 CCITT
  CrcType crcValue;

  cout << "CRC-8 CCITT mit einer Byte-orientierten Lookup Tabelle" <<
      endl << endl;

  if (argc > 1)
  {
    // compute CRC of the passed file (argv[1])
    ifstream f;
    uint8_t* buf;
    unsigned int len;

    try
    {
      f.open(argv[1], ios::in | ios::binary);

      // determine file size
      f.seekg(0, ios::end);
      len = f.tellg();
      f.seekg(0, ios::beg);

      // read file content into byte buffer
      buf = new uint8_t[len];
      f.read((char*)buf, len);
      f.close();

      cout << "Datei:             " << argv[1] << endl;
      cout << "Anzahl Datenbytes: " << len << endl;
```

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

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 profilen.

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

Callgrind: Profilen 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-Profiling 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.

```
#
# Makefile to profile the CRC program with valgrind
# File: makefile
# Gian Danuser, 06.04.2016
#
CC = g++
LINK = g++
# '-g3' to include debugging information
CFLAGS = -c -Wall -pedantic -g3
LFLAGS = -pedantic -Wall
OBJS = Crc.o crcTest.o
EXEC = crcTest

$(EXEC): $(OBJS)
        $(LINK) $(LFLAGS) -o $(@) $(OBJS)

crcTest.o: crcTest.cpp Crc.h
        $(CC) $(CFLAGS) crcTest.cpp

Crc.o: Crc.cpp Crc.h
        $(CC) $(CFLAGS) Crc.cpp

clean:
        rm -f $(EXEC) $(OBJS) callgrind.out.*

callgrind: $(EXEC)
        valgrind --tool=callgrind ./$(EXEC) datafile.txt &&
             kcachegrind
```

```
#
# Makefile to profile the CRC program with valgrind
# File: makefile
# Gian Danuser, 06.04.2016
#
CC = g++
LINK = g++
# '-g3' to include debugging information
CFLAGS = -c -Wall -pedantic -g3 -O3
LFLAGS = -pedantic -Wall
OBJS = Crc.o crcTest.o
EXEC = crcTest

$(EXEC): $(OBJS)
        $(LINK) $(LFLAGS) -o $(@) $(OBJS)

crcTest.o: crcTest.cpp Crc.h
        $(CC) $(CFLAGS) crcTest.cpp

Crc.o: Crc.cpp Crc.h
        $(CC) $(CFLAGS) Crc.cpp

clean:
        rm -f $(EXEC) $(OBJS) callgrind.out.*

callgrind: $(EXEC)
        valgrind --tool=callgrind ./$(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.

```
#
# Makefile to profile the CRC program with valgrind
# File: makefile
# Gian Danuser, 06.04.2016
#
CC = g++
LINK = g++
# '-g3' to include debugging information
CFLAGS = -c -Wall -pedantic -g3
LFLAGS = -pedantic -Wall
OBJS = Crc.o crcTest.o
EXEC = crcTest

$(EXEC): $(OBJS)
        $(LINK) $(LFLAGS) -o $(@) $(OBJS)

crcTest.o: crcTest.cpp Crc.h
        $(CC) $(CFLAGS) crcTest.cpp

Crc.o: Crc.cpp Crc.h
        $(CC) $(CFLAGS) Crc.cpp

clean:
        rm -f $(EXEC) $(OBJS) callgrind.out.*

callgrind: $(EXEC)
        valgrind --tool=callgrind ./$(EXEC) datafile.txt &&
             kcachegrind
```

```
#
# Makefile to profile the CRC program with valgrind
# File: makefile
# Gian Danuser, 06.04.2016
#
CC = g++
LINK = g++
# '-g3' to include debugging information
CFLAGS = -c -Wall -pedantic -g3 -O3
LFLAGS = -pedantic -Wall
OBJS = Crc.o crcTest.o
EXEC = crcTest

$(EXEC): $(OBJS)
        $(LINK) $(LFLAGS) -o $(@) $(OBJS)

crcTest.o: crcTest.cpp Crc.h
        $(CC) $(CFLAGS) crcTest.cpp

Crc.o: Crc.cpp Crc.h
        $(CC) $(CFLAGS) Crc.cpp

clean:
        rm -f $(EXEC) $(OBJS) callgrind.out.*

callgrind: $(EXEC)
        valgrind --tool=callgrind ./$(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 aktu-elle 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 Funk-tion rekursiv aufgerufen wird oder falls ein Pointer auf diese Funktion verwendet wird. Eine weitere Schwie-rigkeit 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 er-zeugt 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. Ver-wenden 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. Verwen-den 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.

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

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 gekenn-zeichnet 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()`):

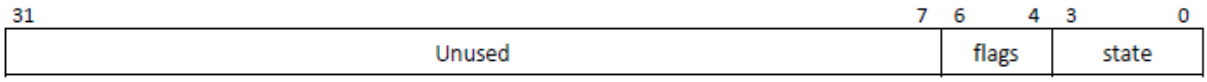
```
(gdb) disassem
Dump of assembler code for function main:
=> 0x0000555555554560 <+0>:   sub    $0x38,%rsp
   0x0000555555554564 <+4>:   movapd 0x214(%rip),%xmm0        # 0x555555554780
   0x000055555555456c <+12>:  lea    0x10(%rsp),%rdi
   0x0000555555554571 <+17>:  mov    %fs:0x28,%rax
   0x000055555555457a <+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



1. Definieren Sie dieses Register mit Hilfe eines Bitfelds. Die Variable (z.B. r) müssen Sie mit volatile kennzeichnen. Wieso?
2. 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.

```
        .file   "main.cpp"
        .text
        .section        .text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        movl    $0, -4(%rsp)
        movl    -4(%rsp), %eax
        orl     $15, %eax
        movl    %eax, -4(%rsp)
        movl    -4(%rsp), %eax
        andl    $-113, %eax
        orl     $48, %eax
        movl    %eax, -4(%rsp)
        movl    -4(%rsp), %eax
        movl    -4(%rsp), %edx
        shrl    $4, %eax
        addl    $1, %eax
        andl    $-113, %edx
        andl    $7, %eax
        sall    $4, %eax
        orl     %edx, %eax
        movl    %eax, -4(%rsp)
        movl    -4(%rsp), %eax
        andl    $96, %eax
        movl    %eax, %edx
        movl    -4(%rsp), %eax
        andl    $-113, %eax
        orl     %edx, %eax
        movl    %eax, -4(%rsp)
        xorl    %eax, %eax
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .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.

```
movl    $0, -4(%rsp)
...
movl    -4(%rsp), %eax
orl     $15, %eax
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).

```
movl    -4(%rsp), %eax
andl    $-113, %eax
orl     $48, %eax
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.

```
movl      -4(%rsp), %eax
movl      -4(%rsp), %edx
shrl      $4, %eax
addl      $1, %eax
andl      $-113, %edx
andl      $7, %eax
sall      $4, %eax
orl%edx,  %eax
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):

```
|63..32|31..16|15-8|7-0|
               |DH.|DL.| <- DH / DL adressieren das High- / Low-Byte der untern 16
                          Bits von Register D
               |DX.....| <- adressiert die unteren 16 Bits von Register D
       |EDX............| <- EDX adressiert die unteren 32 Bits von Register
|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 Operatio-nen 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 ver-wendet 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++, multicols=2]900-Praktika/prak07/Loesung/A3/main.s

```
movl    $0, -4(%rsp)
movl    $15, -4(%rsp)
movl    -4(%rsp), %eax
orl     $48, %eax
movl    %eax, -4(%rsp)
movl    -4(%rsp), %eax
andl    $96, %eax
movl    %eax, -4(%rsp)
```

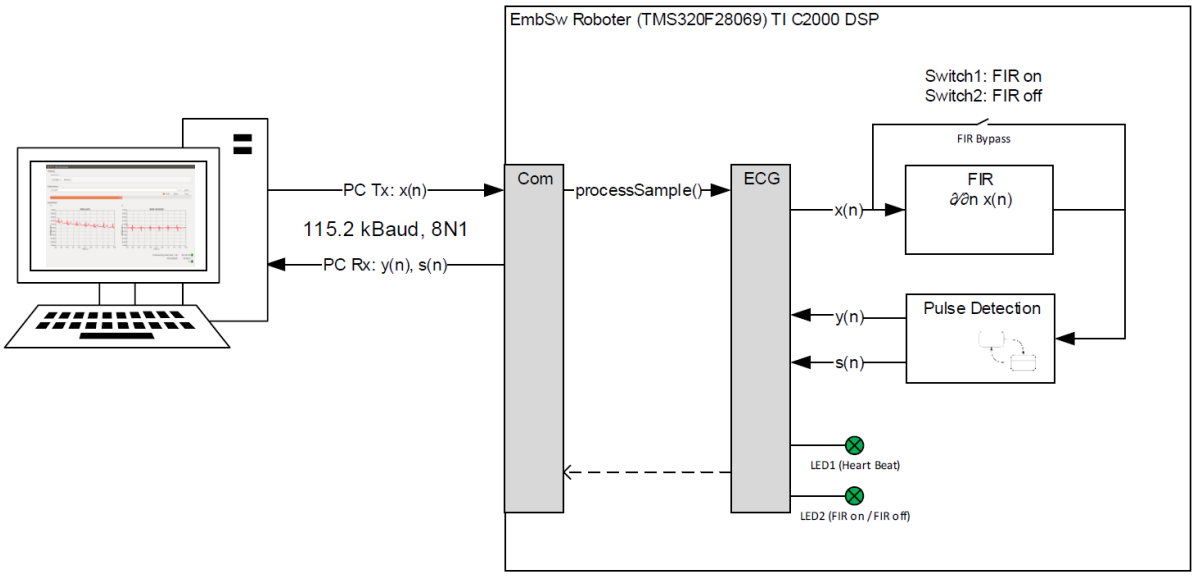
2. Bitmasken verwenden, Bitfelder nicht.

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 Signalflussplan 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 ausgeschalten 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.

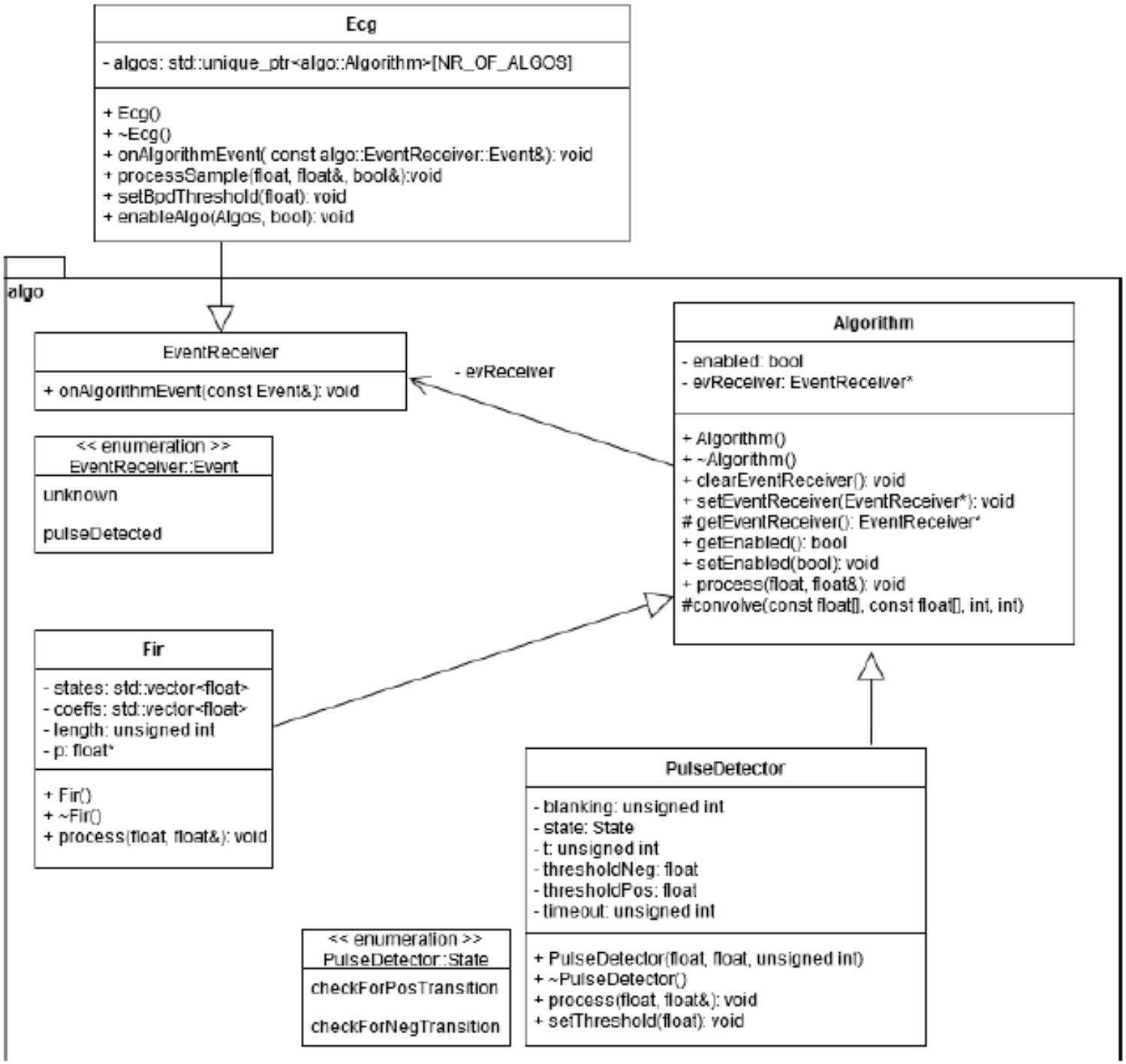


Abbildung 2: Klassendiagramm der EKG

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 ein-fachste Variante wäre ein Filter mit den Koeffizienten $[1 - 1]$, was einem Differenziator entspricht. Das Aus-gangssignal 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 Fen-sterbreite von 5 Samples und eine Ordnung von 3 eine optimale Ableitung generiert. Die resultierenden Ko-effizienten sind unten abgebildet.

```
  static const float sgFilterCoeffs[] = { 0.0833, -0.6667, 0.0000, 0.6667, -0.083
};
```

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. Or-fanidis, 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/algo/-FIR.cpp und Filterkoeffizienten in ./Loesung/Ecg/Ecg.cpp: `const float Ecg::sgFilterCoe = {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-Lib-rary. 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

```
/*
 * Ecg.h
 *
 *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
 */

#ifndef ECG_H_
#define ECG_H_

#include <memory>

#include "algo/EventReceiver.h"
#include "algo/Algorithm.h"

class Ecg : public algo::EventReceiver
{
public:
  ///> Used algorithms
  enum class Algos
  {
    fir = 0, ///< FIR algorithm
    pulseDet ///< pulse detection algorithm
  };

  /**
   * Ctor
   */
        Ecg();

        /**
         * Dtor
         */
        ~Ecg() override = default;

        // overrides EventReceiver::onAlgorithmEvent
        void onAlgorithmEvent(const algo::EventReceiver::Event& ev)
             override;

        /**
         * Enables/disables the algorithm.
         * @param[in] algo algorithm to enable/disable
         * @param[in] en true to enable and false to disable the
              algorithm
         */
        void enableAlgo(Algos algo, bool en);

        /**
         * Sets the balanced photodetector (BPD) threshold.
         * @param[in] value threshold
         */
        void setBpdThreshold(float value);

        /**
         * Processes a sample.
         * @param[in] input input sample
         * @param[out] output output sample
         * @param[out] state pulse detector state (true: new pulse
              detected)
         */
        void processSample(float input, float& output, bool& state);

private:

        enum { NR_OF_ALGOS = static_cast<int>(Algos::pulseDet)+1 };

        static const float sgFilterCoeffs[];
  static const unsigned int sgFilterLength;
```

```
        bool isNewPulseDetected;

        std::unique_ptr<algo::Algorithm> algos[NR_OF_ALGOS];
};

#endif /* ECG_H_ */
```

```
/*
 * Ecg.cpp
 *
 *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
 */

#include "./Ecg.h"

#include <cassert>

#include "algo/FIR.h"
#include "algo/PulseDetector.h"

using algo::FIR;
using algo::PulseDetector;

const float Ecg::sgFilterCoeffs[] =
{
                // TODO: Add the correct filter coefficients to get a robust derivation
                //SOLUTION_BEGIN
                -0.0833,
                0.6667,
                //SOLUTION_END
                0.0,
                //SOLUTION_BEGIN
                -0.6667,
                0.0833,
                //SOLUTION_END
};

const unsigned int Ecg::sgFilterLength = sizeof(sgFilterCoeffs) / sizeof(float);

Ecg::Ecg()
{
        algos[static_cast<int>(Algos::fir)] = std::make_unique<FIR>(sgFilterCoeffs, sgFilterLength );
        algos[static_cast<int>(Algos::pulseDet)] = std::make_unique<PulseDetector>(0.1, -0.1);

        algos[static_cast<int>(Algos::pulseDet)]->setEventReceiver(this);

        algos[static_cast<int>(Algos::fir)]->setEnabled(false);
        algos[static_cast<int>(Algos::pulseDet)]->setEnabled(false);
}

void Ecg::processSample(float input, float& output, bool& state)
{
  isNewPulseDetected = false;
        output = input;
        for (int i = 0; i < NR_OF_ALGOS; ++i)
        {
                if (algos[i] != 0)
                {
                        algos[i]->process(output, output);
                }
        }
        state = isNewPulseDetected;
}

void Ecg::setBpdThreshold(float value)
{
        PulseDetector& pd = *static_cast<PulseDetector*>(algos[static_cast<int>(Algos::pulseDet)].get());
        pd.setThreshold(value);
}

void Ecg::onAlgorithmEvent(const EventReceiver::Event& ev)
{
        switch (ev)
        {
        case EventReceiver::Event::pulseDetected:
          isNewPulseDetected = true;
                break;
        default:
                break;
        }
}

void Ecg::enableAlgo(Algos algo, bool en)
{
        switch (algo)
        {
        case Algos::fir:
                algos[static_cast<int>(Algos::fir)]->setEnabled( en );
                break;
        case Algos::pulseDet:
                algos[static_cast<int>(Algos::pulseDet)]->setEnabled( en );
                break;
        default:
                break;
        }
}
```

```
//
// testEcg.cpp
//
// implements the tests for the Ecg class.
//
// (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
//

#include <iostream>
#include <fstream>
#include <memory>

#include <gtest/gtest.h>

#include "./Ecg.h"

using namespace std;

TEST(EcgTest, DisabledTest)
{
  unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( "ecg.dat" );
  ASSERT_TRUE( fEcgTestData->is_open() );
  istream& in = *fEcgTestData;

  // drop comment lines
  while (in.peek() == '#')
  {
    string s;
    getline(in, s);
  }

  Ecg uut;
  uut.enableAlgo(Ecg::Algos::fir, false);
  uut.enableAlgo(Ecg::Algos::pulseDet, false);

  // store data in buffer and convolve
  float x;
  float y;
  float expY;
  bool state;
  bool expState;
  int cnt = 0;
  while ( in >> x )
  {
    uut.processSample( x, y, state );

    EXPECT_FLOAT_EQ( y, x );
    EXPECT_FALSE( state );
    ++cnt;
  }
  EXPECT_EQ(500, cnt); // check if all samples are processed
}

TEST(EcgTest, FirTest)
{
  unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( "ecg.dat" );
  ASSERT_TRUE( fEcgTestData->is_open() );
  istream& in = *fEcgTestData;

  // drop comment lines
  while (in.peek() == '#')
  {
    string s;
    getline(in, s);
  }

  unique_ptr<ifstream> fExpectedFirOutput = make_unique<ifstream>( "ecgExpected.dat" );
  ASSERT_TRUE( fExpectedFirOutput->is_open() );
  istream& expectedYIn = *fExpectedFirOutput;

  Ecg uut;
  uut.enableAlgo(Ecg::Algos::fir, true);
  uut.enableAlgo(Ecg::Algos::pulseDet, false);

  // store data in buffer and convolve
```

```
  float x;
  float y;
  float expY;
  bool state;
  bool expState;
  int cnt = 0;
  while ( in >> x &&
          expectedYIn >> expY )
  {
    uut.processSample( x, y, state );

    EXPECT_NEAR( y, expY, 1e-6 );
    EXPECT_FALSE( state );
    ++cnt;
  }
  EXPECT_EQ(500, cnt); // check if all samples are processed
}

TEST(EcgTest, PulseDetectorTest)
{
  unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( "ecg.dat" );
  ASSERT_TRUE( fEcgTestData->is_open() );
  istream& in = *fEcgTestData;

  // drop comment lines
  while (in.peek() == '#')
  {
    string s;
    getline(in, s);
  }

  unique_ptr<ifstream> fExpectedFirOutput = make_unique<ifstream>( "
        ecgExpected.dat" );
  ASSERT_TRUE( fExpectedFirOutput->is_open() );
  istream& expectedYIn = *fExpectedFirOutput;

  unique_ptr<ifstream> fExpectedState = make_unique<ifstream>( "
        stateExpected.dat" );
  ASSERT_TRUE( fExpectedState->is_open() );
  istream& expectedStateIn = *fExpectedState;

  Ecg uut;
  uut.enableAlgo(Ecg::Algos::fir, true);
  uut.enableAlgo(Ecg::Algos::pulseDet, true);

  // store data in buffer and convolve
  float x;
  float y;
  float expY;
  bool state;
  bool expState;
  int cnt = 0;
  while ( in >> x &&
          expectedYIn >> expY &&
          expectedStateIn >> expState )
  {
    uut.processSample( x, y, state );

    EXPECT_NEAR( y, expY, 1e-6 );
    EXPECT_EQ( state, expState );
    ++cnt;
  }
  EXPECT_EQ(500, cnt); // check if all samples are processed
}
```

```
/*
 * EventReceiver.h
 *
 *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
 */

#ifndef ALGO_EVENTRECEIVER_H_
#define ALGO_EVENTRECEIVER_H_

namespace algo
{

class EventReceiver
{
public:
  ///> Algorithm event type.
        enum class Event
        {
                unknown, ///< unknown event
                pulseDetected ///< pulse detected event
        };

        /**
         * Is called on each algorithm event.
         * @param ev event
         */
        virtual void onAlgorithmEvent(const Event& ev) = 0;

        /**
         * Dtor
         */
        virtual ~EventReceiver() = default;
};

} /* namespace algo */

#endif /* ALGO_EVENTRECEIVER_H_ */
```

```
/*
 * FIR.h
 *
 *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
 */

#ifndef ALGO_FIR_H_
#define ALGO_FIR_H_

#include <vector>

#include "./Algorithm.h"

namespace algo
{

class FIR : public Algorithm
{
public:
  /**
   * Ctor
   */
        FIR(const float* coeffs, unsigned int length);

        /**
         * Dtor
         */
        ~FIR() override = default;

        // overrides Algorithm::process
        void process(float input, float& output) override;

private:
        std::vector<float> states;
        std::vector<float> coeffs;
        unsigned int length;
        float* p;

};

} /* namespace algo */

#endif /* ALGO_FIR_H_ */
```

```
/*
 * FIR.cpp
 *
 *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
 */

#include "./FIR.h"

namespace algo
{

FIR::FIR(const float* coeffs, unsigned int length) :
    coeffs(coeffs, coeffs+length), // pointer are treated as iterator
    length(length)
{
  states.resize(length, 0.0f);
  p = states.data();
}
```

```
19
20 void FIR::process(float input, float& output)
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
35             in SW
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 */
```

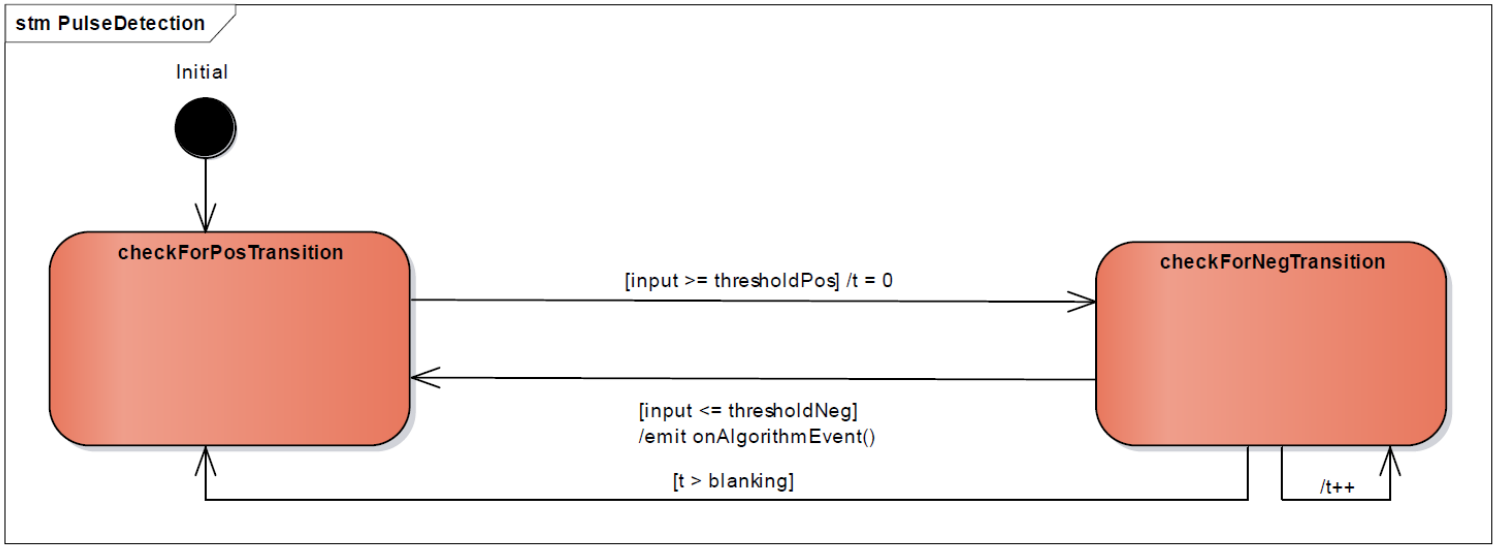
```
1  /*
2   * PulseDetector.h
3   *
4   *  (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5   */
6
7  #ifndef ALGO_PULSEDETECTOR_H_
8  #define ALGO_PULSEDETECTOR_H_
9
10 #include "./Algorithm.h"
11
12 namespace algo
13 {
14
15 class PulseDetector : public Algorithm {
16 public:
17
18     /**
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         void process(float input, float& output) override;
33
34         /**
35          * Sets the pulse detection threshold.
36          * @param[in] value pulse detection threshold
37          */
38         void setThreshold(float value);
39
40 private:
41
42         float thresholdNeg;
43         float thresholdPos;
44
45   // TODO: Private declarations
46   //SOLUTION_BEGIN
47         enum class State
48         {
49                 checkForNegTransition,
50                 checkForPosTransition
51         };
52
53         unsigned int blanking;
54         unsigned int t;
55
56         State state;
57   //SOLUTION_END
58 };
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         //SOLUTION_BEGIN
30         output = input;
31         if (getEnabled())
32         {
33     switch (state)
34     {
35       case State::checkForPosTransition:
36         if (input >= thresholdPos )
37         {
38           t = 0;
39           state = State::checkForNegTransition;
40         }
41         break;
42       case State::checkForNegTransition:
43         if (input <= thresholdNeg)
44         {
45           if ( getEventReceiver() ) {
46             getEventReceiver()->onAlgorithmEvent(EventReceiver::Event::
                   pulseDetected);
47           }
48           state = State::checkForPosTransition;
49         }
50         else if (t >= blanking)
51         {
52           state = State::checkForPosTransition;
53         }
54         break;
55     }
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 Ein-gangssignal 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-Mecha-nismus 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 zeitli-ches 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 über-schritten 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

```cpp
/*
 * Algorithm.h
 *
 *  Created on: 08.04.2015
 *      Author: phoerler
 */

#ifndef ALGOS_ALGORITHM_H_
#define ALGOS_ALGORITHM_H_

#include "./EventReceiver.h"

namespace algo
{

class Algorithm
{
  public:
    /**
     * Ctor
     */
    Algorithm() :
        enabled(false),
        evReceiver( nullptr )
    {
    }

    /**
     * Dtor
     */
    virtual ~Algorithm() = default;

    /**
     * Processing function of the algorithm. Normally, this function will be overwritten by the derived classes, otherwise the input is passed to the output directly.
     * @param[in] input input sample
     * @param[out] output output sample
     */
    virtual void process(float input, float& output)
    {
      output = input;
    }

    /**
     * Enable the algorithm
     * @param[in] val enable or disable the algorithm
     */
    void setEnabled(bool val)
    {
      enabled = val;
    }

    /**
     * Returns whether the algorithm is enabled or not.
     * @return returns whether the algorithm is enabled or not
     */
    bool getEnabled() const
    {
      return enabled;
    }

    /**
     * Stores the given object as receiver for algorithm events.
     * @param[in] receiver reference to the object which is called if an algorithm event occurred
     */
    void setEventReceiver(EventReceiver* receiver)
    {
      evReceiver = receiver;
    }

    /**
     * Removes the event receiver.
     */
    void clearEventReceiver()
    {
      evReceiver = nullptr;
    }

  protected:
    EventReceiver* getEventReceiver() const
    {
      return evReceiver;
    }

    /**
     * Convolves input buffer and filter with a given length at a running index.
     * @param[in] filter filter coefficients of length lf
     * @param[in] buffer sample buffer of length lf
     * @param[in] lf length of the sample and filter coefficient buffers
     * @return returns the result of the discrete convolution
     */
    static float convolve( const float filter[],
                           const float buffer[],
                           int lf,
                           int pos )
    {
      float sum = 0;
      int k;
      for (k = 0; k < lf; ++k)
      {
        sum += filter[k] * buffer[(lf + pos - k) % lf];
      }
      return sum;
    }

  private:
    bool enabled;
    EventReceiver* evReceiver;
};

} /* namespace algo */

#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 Elementfunktio-nen sind einfach vorhanden und werden mit call aufgerufen. Bei Optimierungstufe 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

```
/*
 * Stack.h
 *
 *  Created on: 24.04.2015
 *      Author: rbondere
 */

#ifndef STACK_H_
#define STACK_H_

template<typename ElemType, int size = 10>
class Stack
{
  public:
    Stack();
    // Default-Konstruktor

    void push(const ElemType& e);
    // legt ein Element auf den Stack, falls der Stack noch nicht voll ist
    // wasError() gibt Auskunft, ob push() erfolgreich war

    ElemType pop();
    // nimmt ein Element vom Stack, falls der Stack nicht leer ist
    // wasError() gibt Auskunft, ob pop() erfolgreich war

    ElemType peek() const;
    // liest das oberste Element vom Stack, falls der Stack nicht leer ist
    // wasError() gibt Auskunft, ob peek() erfolgreich war

    bool isEmpty() const {return top == 0;}
    // return: true: Stack ist leer
    //         false: sonst

    bool isFull() const;
    // return: true: Stack ist voll
    //         false: sonst

    bool wasError() const {return error;}
    // return: true: Operation war fehlerhaft
    //         false: sonst

  private:
    ElemType elems[size]; // Speicher fuer Speicherung des Stacks
    int top;              // Arrayindex des naechsten freien Elements
    mutable bool error;   // true: Fehler passiert; false: sonst
    // mutable: auch const-Methoden koennen dieses Attribut setzen
};

// ugly include
#include "Stack.cpp"
#endif // STACK_H_
```

```cpp
/*
 * Stack.cpp
 *
 *  Created on: 24.04.2015
 *      Author: rbondere
 */


template<typename ElemType, int size>
Stack<ElemType, size>::Stack() :
  top(0), error(false)
{
}

template<typename ElemType, int size>
void Stack<ElemType, size>::push(const ElemType& e)
{
  error = isFull();
  if (!error)
  {
    elems[top] = e;
    ++top;
  }
}

template<typename ElemType, int size>
ElemType Stack<ElemType, size>::pop()
{
  error = isEmpty();
  if (!error)
  {
    --top;
    return elems[top];
  }
  else
    return 0;
}

template<typename ElemType, int size>
ElemType Stack<ElemType, size>::peek() const
{
  error = isEmpty();
  if (!error)
    return elems[top - 1];
  else
    return 0;
}

template<typename ElemType, int size>
bool Stack<ElemType, size>::isFull() const
{
  return top == size;
}
```

```cpp
/*
 * StackTest.cpp
 *
 *  Created on: 09.05.2012
 *      Author: rbondere
 */

#include "StackUI.h"
using namespace std;

enum
{
  stackSize = 3
};

int main(void)
{
  StackUI<int, stackSize> sUI;
  sUI.dialog();
  StackUI<int, 4> sUI4;
  sUI4.dialog();

  return 0;
}
```

```cpp
/*
 * StackUI.h
 *
 * User Interface for Stack application
 * Created on: 09.05.2012
 *      Author: rbondere
 */

#ifndef STACKUI_H_
#define STACKUI_H_
#include "Stack.h"

template<typename ElemType, int size = 10>
class StackUI
{
  public:
    void dialog();
    // starts the user dialog
  private:
    Stack<ElemType, size> s;
};

// ugly include
#include "StackUI.cpp"

#endif /* STACKUI_H_ */
```

```cpp
/*
 * StackUI.cpp
 *
 *  Created on: 09.05.2012
 *      Author: rbondere
 */

#include <iostream>

// DO NOT #include "StackUI.h" !!
using namespace std;

template<typename ElemType, int size>
void StackUI<ElemType, size>::dialog()
{
  char ch = 0;
  ElemType e;
  do
  {
    cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty) ";
    cin >> ch;
    switch (ch)
    {
      case 'q':
      case 'Q': // quit
        break;
      case 'u':
      case 'U': // push
        cout << "\nElement to push: ";
        cin >> e;
        s.push(e);
        if (s.wasError())
          cout << "\nError: Stack full.";
        break;
      case 'o':
      case 'O': // pop
        e = s.pop();
        if (s.wasError())
          cout << "\nError: Stack is empty (nothing to pop).";
        else
          cout << "\nPopped element " << e;
        break;
      case 'k':
      case 'K': // peek
        e = s.peek();
        if (s.wasError())
```

```
47          cout << "\nError: Stack is empty (nothing to peek).";
48        else
49          cout << "\nPeeked element " << e;
50        break;
51      case 'e':
52      case 'E': // isEmpty
53        if (s.isEmpty())
54          cout << "\nStack is empty.";
55        else
56          cout << "\nStack contains elements.";
57        break;
58      default:
59        cout << "\nInvalid operation.";
60        break;
61    }
62  } while (ch != 'Q' && ch != 'q');
63
64}
```

Wenn ohne Optimierung compiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen in-klusive 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  0000000000000f90 W Stack<int, 3>::pop()
5  0000000000000f1e W Stack<int, 3>::push(int const&)
6  0000000000000ab2 W Stack<int, 3>::Stack()
7  0000000000000ab2 W Stack<int, 3>::Stack()
8  00000000000010c2 W Stack<int, 4>::pop()
9  0000000000001050 W Stack<int, 4>::push(int const&)
10 0000000000000ce8 W Stack<int, 4>::Stack()
11 0000000000000ce8 W Stack<int, 4>::Stack()
12 0000000000000ad0 W StackUI<int, 3>::dialog()
13 0000000000000a7a W StackUI<int, 3>::StackUI()
14 0000000000000a7a W StackUI<int, 3>::StackUI()
15 0000000000000d06 W StackUI<int, 4>::dialog()
16 0000000000000a96 W StackUI<int, 4>::StackUI()
17 0000000000000a96 W StackUI<int, 4>::StackUI()
18 0000000000000fec W Stack<int, 3>::peek() const
19 0000000000001182 W Stack<int, 3>::isFull() const
20 000000000000103a W Stack<int, 3>::isEmpty() const
21 0000000000000f7e W Stack<int, 3>::wasError() const
22 000000000000111e W Stack<int, 4>::peek() const
23 000000000000119a W Stack<int, 4>::isFull() const
24 000000000000116c W Stack<int, 4>::isEmpty() const
25 00000000000010b0 W Stack<int, 4>::wasError() const
26 ...
27
28
29 Build mit -O3:
30 $ nm -C StackTest
31 ...
32 0000000000000a80 W StackUI<int, 3>::dialog()
33 0000000000000d50 W StackUI<int, 4>::dialog()
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::di` 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 _ZN7StackUIIiLi3EE6dialogEv
2  ...
3  call _ZN7StackUIIiLi4EE6dialogEv
```

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 Elementfunktio-nen isEmpty() und isFull() unabhängig von der Grösse oder nicht? Weiss man das bereits bei der De-klaration oder ist es erst bei der Implementation bekannt? Aus der Implementation geht hervor, dass die Me-thode 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 zugrei-fen) 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 compiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen und Konstruktoren der Basisklasse sind nur noch einfach vorhanden, alle anderen Elementfunktionen sind dop-pelt vorhanden . Bei älteren Compilerversionen (z.B. g++-Version v3.4.4) waren mit Optimierungstufe 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 nicht voll
       ist
21     // wasError() gibt Auskunft, ob push() erfolgreich war
22
23     bool isFull() const;
24     // return: true: Stack ist voll
25     //         false: sonst
26
27   private:
28     ElemType elems[size]; // Speicher fuer Speicherung des Stacks
29 };
30
31 // ugly include
32 #include "Stack.cpp"
33 #endif // STACK_H_
```

```cpp
/*
 * Stack.cpp
 *
 *  Created on: 24.04.2015
 *      Author: rbondere
 */


template<typename ElemType, int size>
Stack<ElemType, size>::Stack() :
  StackNoSize<ElemType>(&elems[0])
{
}

template<typename ElemType, int size>
void Stack<ElemType, size>::push(const ElemType& e)
{
  StackNoSize<ElemType>::error = isFull(); // Compiler kennt error nicht ohne Class-Qualifier
  if (!StackNoSize<ElemType>::error)
  {
    elems[StackNoSize<ElemType>::top] = e; // dito top
    ++StackNoSize<ElemType>::top;
  }
}

template<typename ElemType, int size>
bool Stack<ElemType, size>::isFull() const
{
  return StackNoSize<ElemType>::top == size;
}
```

```cpp
/*
 * StackNoSize.h
 *
 *  Created on: 09.05.2012
 *      Author: rbondere
 */

#ifndef STACKNOSIZE_H_
#define STACKNOSIZE_H_

template<typename ElemType>
class StackNoSize
{
  public:
    ElemType pop();
    // nimmt ein Element vom Stack, falls der Stack nicht leer ist
    // wasError() gibt Auskunft, ob pop() erfolgreich war

    ElemType peek() const;
    // liest das oberste Element vom Stack, falls der Stack nicht leer ist
    // wasError() gibt Auskunft, ob peek() erfolgreich war

    bool isEmpty() const {return top == 0;}
    // return: true: Stack ist leer
    //         false: sonst

    bool wasError() const {return error;}
    // return: true: Operation war fehlerhaft
    //         false: sonst

  protected:
    StackNoSize(ElemType* pArr = 0);
    // Default-Konstruktor (nur Unterklasen sollen Objekte gruenden koennen)
    int top;              // Arrayindex des naechsten freien Elements
    mutable bool error;   // true: Fehler passiert; false: sonst
    // mutable: auch const-Methoden koennen dieses Attribut setzen

  private:
    ElemType* pElems;     // Pointer auf Array in Unterklasse (nicht ganz sauber)
};

// ugly include
#include "StackNoSize.cpp"
#endif // STACKNOSIZE_H_
```

```cpp
/*
 * StackNoSize.cpp
 *
 *  Created on: 24.04.2015
 *      Author: rbondere
 */


template<typename ElemType>
StackNoSize<ElemType>::StackNoSize(ElemType* pArr) :
  top(0), error(false), pElems(pArr)
{
}

template<typename ElemType>
ElemType StackNoSize<ElemType>::pop()
{
  error = isEmpty();
  if (!error)
  {
    --top;
    return pElems[top];
  }
  else
    return ElemType(0);
}

template<typename ElemType>
ElemType StackNoSize<ElemType>::peek() const
{
  error = isEmpty();
  if (!error)
    return pElems[top - 1];
  else
    return ElemType(0);
}
```

```cpp
/*
 * StackTest.cpp
 *
 *  Created on: 09.05.2012
 *      Author: rbondere
 */

#include "StackUI.h"
using namespace std;

enum
{
  stackSize = 3
};

int main(void)
{
  StackUI<int, stackSize> sUI;
  sUI.dialog();
  StackUI<int, 4> sUI4;
  sUI4.dialog();

  return 0;
}
```

```cpp
/*
 * StackUI.h
 *
 * User Interface for Stack application
 *
 * Created on: 09.05.2012
 *      Author: rbondere
 */

```

```
#ifndef STACKUI_H_
#define STACKUI_H_
#include "Stack.h"

template<typename ElemType, int size = 10>
class StackUI
{
  public:
    void dialog();
    // starts the user dialog
  private:
    Stack<ElemType, size> s;
};

// ugly include
#include "StackUI.cpp"

#endif /* STACKUI_H_ */
```

```
/*
 * StackUI.cpp
 *
 *  Created on: 09.05.2012
 *      Author: rbondere
 */

#include <iostream>

// DO NOT #include "StackUI.h" !!
using namespace std;

template<typename ElemType, int size>
void StackUI<ElemType, size>::dialog()
{
  char ch = 0;
  ElemType e;
  do
  {
    cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty) ";
    cin >> ch;
    switch (ch)
    {
      case 'q':
      case 'Q': // quit
        break;
      case 'u':
      case 'U': // push
        cout << "\nElement to push: ";
        cin >> e;
        s.push(e);
        if (s.wasError())
          cout << "\nError: Stack full.";
        break;
      case 'o':
      case 'O': // pop
        e = s.pop();
        if (s.wasError())
          cout << "\nError: Stack is empty (nothing to pop).";
        else
          cout << "\nPopped element " << e;
        break;
      case 'k':
      case 'K': // peek
        e = s.peek();
        if (s.wasError())
          cout << "\nError: Stack is empty (nothing to peek).";
        else
          cout << "\nPeeked element " << e;
        break;
      case 'e':
      case 'E': // isEmpty
        if (s.isEmpty())
          cout << "\nStack is empty.";
        else
          cout << "\nStack contains elements.";
        break;
      default:
        cout << "\nInvalid operation.";
        break;
    }
  } while (ch != 'Q' && ch != 'q');

}
```

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 imple-mentieren. Sie müssen die Klassen `PoolAllocator, HeapException, HeapSizeMismatc` `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 Elementfunk-tionen 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`.

```
template<std::size_t heapSize, std::size_t elemSize>
class PoolAllocator
{
    public:
        PoolAllocator(/* TODO: params (heap address */);
        void* allocate(std::size_t bytes);
        // TODO: implement this
        // throw a HeapSizeMismatch Exception if 'bytes' doesn't match elemSize
        // throw a OutOfHeap Exception if requested 'bytes' aren't available
        void deallocate(/* TODO: params */) noexcept;
        // TODO: implement this
        // add this element to the freelist
        // set the pointer to nullptr
    private:
        union Node
```

```
        {
            uint8_t data[elemSize]; // sizeof(data) should be >= sizeof(Node*)
            Node* next;
        };
        Node* freeList;
    };
    template<std::size_t heapSize, std::size_t elemSize>
    PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t* heapAddr)
    {
    // TODO: implement this
}
```

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

```
p[0] = 0x7ffe93438070
p[1] = 0x7ffe93438078
p[2] = 0x7ffe93438080
p[1] = 0
p[3] = 0x7ffe93438078
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.

```
//
     ================================================================
// Name         : FixedPool.cpp
// Author       : Reto Bonderer
// Version      : 02.06.2016
// Description : Fixed-size pool implementation
//
     ================================================================


#include <iostream>
#include "PoolAllocator.h"
using namespace std;
using namespace dynamicMemory;

int main()
{
  enum
  {
    testHeapSize = 200, testElemSize = 8,
    nElems = testHeapSize / testElemSize,
    nPtrs = 80
  };
  static uint8_t buffer[testHeapSize]; // our heap
  void* p[nPtrs] = { nullptr }; // some void-pointers
  PoolAllocator<testHeapSize, testElemSize> pool(&buffer[0]);

  try
  {
    p[0] = pool.allocate(testElemSize);
    cout << "p[0] = " << hex << p[0] << endl;
    p[1] = pool.allocate(testElemSize);
    cout << "p[1] = " << hex << p[1] << endl;
    p[2] = pool.allocate(testElemSize);
    cout << "p[2] = " << hex << p[2] << endl;
    pool.deallocate(p[1]);
    cout << "p[1] = " << hex << p[1] << endl;
    p[3] = pool.allocate(testElemSize);
    cout << "p[3] = " << hex << p[3] << endl; // should be former
         address of p[1]
    p[4] = pool.allocate(testElemSize - 1);
    cout << "p[4] = " << hex << p[4] << endl; // should throw
         HeapSizeMismatch exception
  }
  catch (const HeapSizeMismatch& e)
  {
    cout << "Heapsize mismatch exception occurred" << endl;
  }
  catch (const OutOfHeap& e)
  {
    cout << "Out of heap exception occurred" << endl;
  }
  catch (...)
  {
    cout << "Shouldn't get here" << endl;
  }

  try
  {
    for (int i = 5; i < nElems + 2; ++i)  // there are 3 objects so
         far => (nElems-3) available
      p[i] = pool.allocate(testElemSize);
    cout << dec << "p[" << nElems + 1 << "] = " << hex << p[nElems +
         1] << endl;
    cout << dec << "p[" << nElems + 2 << "] = " << hex << p[nElems +
         2] << endl; // = 0
    p[nElems + 3] = pool.allocate(testElemSize); // should throw
         OutOfHeap exception
  }
  catch (const HeapSizeMismatch& e)
  {
    cout << "Heapsize mismatch exception occurred" << endl;
  }
  catch (const OutOfHeap& e)
  {
    cout << "Out of heap exception occurred" << endl;
  }
  catch (...)
  {
    cout << "Shouldn't get here" << endl;
  }

  cout << "Deallocate all";
  for (int i = 0; i < nPtrs; ++i)
  {
    try
    {
      pool.deallocate(p[i]);
      cout << ".";
    }
    catch (...)
    {
      cout << "Shouldn't get here" << endl;
    }
  }
  return 0;
}
```

```cpp
/*
 * PoolAllocator.h
 *
 *  Created on: 27.04.2020
 *      Author: Reto Bonderer
 */

#ifndef POOLALLOCATOR_H_
#define POOLALLOCATOR_H_

#include <stdint.h>

namespace dynamicMemory
{
class HeapException
{
};

class HeapSizeMismatch: public HeapException
{
};

class OutOfHeap: public HeapException
{
};

template<std::size_t heapSize, std::size_t elemSize>
class PoolAllocator
{
  public:
    PoolAllocator(uint8_t* heapAddr);
    void* allocate(std::size_t bytes);
    void deallocate(void*& ptr) noexcept; // void*& allows writing ptr
  private:
    union Node
    {
      uint8_t data[elemSize];
      Node* next;
    };
    Node* freeList;
};

template<std::size_t heapSize, std::size_t elemSize>
PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t* heapAddr) :
    freeList(reinterpret_cast<Node*>(heapAddr))
{
  const std::size_t nElems = heapSize / sizeof(Node);
  for (std::size_t i = 0; i < nElems - 1; ++i) // link array elements together
    freeList[i].next = &freeList[i + 1];
  freeList[nElems - 1].next = nullptr;
}

template<std::size_t heapSize, std::size_t elemSize>
void* PoolAllocator<heapSize, elemSize>::allocate(std::size_t bytes)
{
  if (bytes != elemSize)
    throw HeapSizeMismatch();
  if (freeList != nullptr)
  {
    void* pMem = freeList;
    freeList = freeList->next;
    return pMem;
  }
  else
    throw OutOfHeap();
}

template<std::size_t heapSize, std::size_t elemSize>
void PoolAllocator<heapSize, elemSize>::deallocate(void*& ptr) noexcept
{
  if (ptr == nullptr)
    return;
  Node* p = static_cast<Node*>(ptr);
  p->next = freeList;
  freeList = p;
  ptr = nullptr;  // increases safety
}

} /* namespace dynamicMemory */

#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();`

```cpp
//
//============================================================================
// Name        : FixedPoolCeil.cpp
// Author      : Reto Bonderer
// Version     : 02.06.2016
// Description : Fixed-size pool with upper limit implementation
//
//============================================================================

#include <iostream>
#include "PoolAllocator.h"
using namespace std;
using namespace dynamicMemory;

int main()
{
  enum
  {
    testHeapSize = 200, testElemSize = 8,
    nElems = testHeapSize / testElemSize,
    nPtrs = 80
  };
  static uint8_t buffer[testHeapSize]; // our heap
  void* p[nPtrs] = { nullptr }; // some void-pointers

  PoolAllocator<testHeapSize, testElemSize> pool(&buffer[0]);
  try
  {
    p[0] = pool.allocate(testElemSize);
    cout << "p[0] = " << hex << p[0] << endl;
    p[1] = pool.allocate(testElemSize);
    cout << "p[1] = " << hex << p[1] << endl;
    p[2] = pool.allocate(testElemSize - 1);                 // should work
    cout << "p[2] = " << hex << p[2] << endl;
    pool.deallocate(p[1]);
    cout << "p[1] = " << hex << p[1] << endl;
    p[3] = pool.allocate(testElemSize);
    cout << "p[3] = " << hex << p[3] << endl; // should be former address of p[1]
    p[4] = pool.allocate(testElemSize + 1);
    cout << "p[4] = " << hex << p[4] << endl; // should throw HeapSizeMismatch exception
  }
  catch (const HeapSizeMismatch& e)
  {
    cout << "Heapsize mismatch exception occurred" << endl;
  }
  catch (const OutOfHeap& e)
  {
    cout << "Out of heap exception occurred" << endl;
  }
  catch (...)
  {
    cout << "Shouldn't get here" << endl;
  }

  try
  {
    for (int i = 5; i < nElems + 2; ++i)  // there are 3 objects so far => (nElems-3) available
      p[i] = pool.allocate(testElemSize);
    cout << dec << "p[" << nElems + 1 << "] = " << hex << p[nElems + 1] << endl;
    cout << dec << "p[" << nElems + 2 << "] = " << hex << p[nElems + 2] << endl; // = 0
    p[nElems + 3] = pool.allocate(testElemSize); // should throw OutOfHeap exception
  }
  catch (const HeapSizeMismatch& e)
  {
    cout << "Heapsize mismatch exception occurred" << endl;
  }
  catch (const OutOfHeap& e)
  {
    cout << "Out of heap exception occurred" << endl;
  }
  catch (...)
  {
    cout << "Shouldn't get here" << endl;
```

```
  }

  cout << "Deallocate all";
  for (int i = 0; i < nPtrs; ++i)
  {
    try
    {
      pool.deallocate(p[i]);
      cout << ".";
    }
    catch (...)
    {
      cout << "Shouldn't get here" << endl;
    }
  }
  return 0;
}
```

```
/*
 * PoolAllocator.h
 *
 *  Created on: 27.04.2020
 *      Author: Reto Bonderer
 */

#ifndef POOLALLOCATOR_H_
#define POOLALLOCATOR_H_

#include <stdint.h>

namespace dynamicMemory
{
class HeapException
{
};

class HeapSizeMismatch: public HeapException
{
};

class OutOfHeap: public HeapException
{
};

template<std::size_t heapSize, std::size_t elemSize>
class PoolAllocator
{
  public:
    PoolAllocator(uint8_t* heapAddr);
    void* allocate(std::size_t bytes);
    void deallocate(void*& ptr) noexcept; // void*& allows writing ptr
  private:
    union Node
    {
      uint8_t data[elemSize];
      Node* next;
    };
    Node* freeList;
};

template<std::size_t heapSize, std::size_t elemSize>
PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t* heapAddr) :
    freeList(reinterpret_cast<Node*>(heapAddr))
{
  const std::size_t nElems = heapSize / sizeof(Node);
  for (std::size_t i = 0; i < nElems - 1; ++i) // link array elements together
    freeList[i].next = &freeList[i + 1];
  freeList[nElems - 1].next = nullptr;
}

template<std::size_t heapSize, std::size_t elemSize>
void* PoolAllocator<heapSize, elemSize>::allocate(std::size_t bytes)
{
  if (bytes > elemSize)
    throw HeapSizeMismatch();
  if (freeList != nullptr)
  {
    void* pMem = freeList;
    freeList = freeList->next;
    return pMem;
  }
  else
    throw OutOfHeap();
}

template<std::size_t heapSize, std::size_t elemSize>
void PoolAllocator<heapSize, elemSize>::deallocate(void*& ptr)
    noexcept
{
  if (ptr == nullptr)
    return;
  Node* p = static_cast<Node*>(ptr);
  p->next = freeList;
  freeList = p;
  ptr = nullptr;  // increases safety
}

} /* namespace dynamicMemory */

#endif /* POOLALLOCATOR_H_ */
```

10.2 Aufgabe 2: Block Allocator

Implementieren Sie nun einen Block Allocator. Die Klasse `BlockAllocator` soll aus 4 Fixed-size Pools ge-mäss 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.

```
//
      ============================================================================
// Name        : BlockAllocation.cpp
// Author      : Reto Bonderer
// Version     : 02.06.2016
// Description : Block allocation with 4 fixed-size pools
//
      ============================================================================


#include <iostream>
#include "BlockAllocator.h"
using namespace std;
using namespace dynamicMemory;

int main()
{
  enum
  {
    testPool1Size = 200, testElem1Size = 4,
    testPool2Size = 200, testElem2Size = 8,
    testPool3Size = 512, testElem3Size = 32,
    testPool4Size = 1024, testElem4Size = 128,
    nPtrs = 20
  };
  static uint8_t pool1Heap[testPool1Size];
  static uint8_t pool2Heap[testPool2Size];
  static uint8_t pool3Heap[testPool3Size];
  static uint8_t pool4Heap[testPool4Size];

  BlockAllocator<testPool1Size, testElem1Size,
                 testPool2Size, testElem2Size,
                 testPool3Size, testElem3Size,
                 testPool4Size, testElem4Size> block(&pool1Heap[0], &
                      pool2Heap[0],
                                                     &pool3Heap[0], &
                                                          pool4Heap[0]);

  void* p[nPtrs] = { nullptr }; // some void-pointers
  cout << "Memory Map" << endl;
  cout << "Pool 1: " << hex << (void*)pool1Heap << "-"
       << (void*)(pool1Heap + testPool1Size - 1) << " size: " << dec
       << testPool1Size << endl;
```

```
  cout << "Pool 2: " << hex << (void*)pool2Heap << "-"
       << (void*)(pool2Heap + testPool2Size - 1) << " size: " << dec
       << testPool2Size << endl;
  cout << "Pool 3: " << hex << (void*)pool3Heap << "-"
       << (void*)(pool3Heap + testPool3Size - 1) << " size: " << dec
       << testPool3Size << endl;
  cout << "Pool 4: " << hex << (void*)pool4Heap << "-"
       << (void*)(pool4Heap + testPool4Size - 1) << " size: " << dec
       << testPool4Size << endl;

  try  // ... some allocations
  {
    p[0] = block.allocate(4);
    cout << "p[0] = " << hex << p[0] << endl;
    p[1] = block.allocate(4);
    cout << "p[1] = " << hex << p[1] << endl;
    p[2] = block.allocate(3); // should work
    cout << "p[2] = " << hex << p[2] << endl;
    block.deallocate(p[1]);
    cout << "p[1] = " << hex << p[1] << endl;
    p[3] = block.allocate(4);
    cout << "p[3] = " << hex << p[3] << endl; // should be former address
         of p[1]
    p[4] = block.allocate(128);
    cout << "p[4] = " << hex << p[4] << endl;
    p[5] = block.allocate(140);
    cout << "p[5] = " << hex << p[5] << endl; // should throw
         HeapSizeMismatch exception
    p[6] = block.allocate(123);
    cout << "p[6] = " << hex << p[6] << endl;
  }
  catch (const HeapSizeMismatch& e)
  {
    cout << "Heapsize mismatch exception occurred" << endl;
  }
  catch (const OutOfHeap& e)
  {
    cout << "Out of heap exception occurred" << endl;
  }
  catch (...)
  {
    cout << "Shouldn't get here" << endl;
  }

  cout << "Deallocate all";
  for (int i = 0; i < nPtrs; ++i)
  {
    try
    {
      block.deallocate(p[i]);
      cout << ".";
    }
    catch (...)
    {
      cout << "Shouldn't get here" << endl;
    }
  }
  return 0;
}
```

```
/*
 * PoolAllocator.h
 *
 *  Created on: 27.04.2020
 *      Author: Reto Bonderer
 */

#ifndef POOLALLOCATOR_H_
#define POOLALLOCATOR_H_

#include <stdint.h>

namespace dynamicMemory
{
class HeapException
{
};

class HeapSizeMismatch: public HeapException
{
};

class OutOfHeap: public HeapException
{
};

template<std::size_t heapSize, std::size_t elemSize>
class PoolAllocator
{
  public:
    PoolAllocator(uint8_t* heapAddr);
    void* allocate(std::size_t bytes);
    void deallocate(void*& ptr) noexcept; // void*& allows writing ptr
  private:
    union Node
    {
      uint8_t data[elemSize];
      Node* next;
    };
    Node* freeList;
};

template<std::size_t heapSize, std::size_t elemSize>
PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t* heapAddr) :
    freeList(reinterpret_cast<Node*>(heapAddr))
{
  const std::size_t nElems = heapSize / sizeof(Node);
  for (std::size_t i = 0; i < nElems - 1; ++i) // link array elements
       together
    freeList[i].next = &freeList[i + 1];
  freeList[nElems - 1].next = nullptr;
}

template<std::size_t heapSize, std::size_t elemSize>
void* PoolAllocator<heapSize, elemSize>::allocate(std::size_t bytes)
{
  if (bytes > elemSize)
    throw HeapSizeMismatch();
  if (freeList != nullptr)
  {
    void* pMem = freeList;
    freeList = freeList->next;
    return pMem;
  }
  else
    throw OutOfHeap();
}

template<std::size_t heapSize, std::size_t elemSize>
void PoolAllocator<heapSize, elemSize>::deallocate(void*& ptr) noexcept
{
  if (ptr == nullptr)
    return;
  Node* p = static_cast<Node*>(ptr);
  p->next = freeList;
  freeList = p;
  ptr = nullptr;  // increases safety
}

} /* namespace dynamicMemory */

#endif /* POOLALLOCATOR_H_ */
```

```
/*
 * BlockAllocator.h
 *
 *  Created on: 27.04.2020
 *      Author: Reto Bonderer
 */

#ifndef BLOCKALLOCATOR_H_
#define BLOCKALLOCATOR_H_

#include "PoolAllocator.h"

namespace dynamicMemory
{

template<std::size_t pool1Size, std::size_t elem1Size,  // elem1Size <=
     elem2Size <= ...
         std::size_t pool2Size, std::size_t elem2Size,  // elem3Size <=
              elem4Size!!
         std::size_t pool3Size, std::size_t elem3Size,
         std::size_t pool4Size, std::size_t elem4Size>
class BlockAllocator
{
  public:
    BlockAllocator(uint8_t* pool1Heap,
                   uint8_t* pool2Heap,
                   uint8_t* pool3Heap,
                   uint8_t* pool4Heap) :
           pool1(pool1Heap),
           pool2(pool2Heap),
           pool3(pool3Heap),
           pool4(pool4Heap),
           pool1Addr(pool1Heap),
           pool2Addr(pool2Heap),
           pool3Addr(pool3Heap),
           pool4Addr(pool4Heap)
```

```
    {
    }
    void* allocate(std::size_t bytes);
    void deallocate(void*& ptr) noexcept;

  private:
    PoolAllocator<pool1Size, elem1Size> pool1;
    PoolAllocator<pool2Size, elem2Size> pool2;
    PoolAllocator<pool3Size, elem3Size> pool3;
    PoolAllocator<pool4Size, elem4Size> pool4;
    uint8_t* pool1Addr;
    uint8_t* pool2Addr;
    uint8_t* pool3Addr;
    uint8_t* pool4Addr;
};

template<std::size_t pool1Size, std::size_t elem1Size,
         std::size_t pool2Size, std::size_t elem2Size,
         std::size_t pool3Size, std::size_t elem3Size,
         std::size_t pool4Size, std::size_t elem4Size>
void* BlockAllocator<pool1Size, elem1Size, pool2Size, elem2Size,
                     pool3Size, elem3Size, pool4Size, elem4Size>::
                          allocate(std::size_t bytes)
{
  void* p;
  try
  {
    p = pool1.allocate(bytes);
  }
  catch (const HeapException& e)
  {
    try
    {
      p = pool2.allocate(bytes);
    }
    catch (const HeapException& e)
    {
      try
      {
        p = pool3.allocate(bytes);
      }
      catch (const HeapException& e)
      {
        p = pool4.allocate(bytes);
      }
    }
  }
  return p;
}

template<std::size_t pool1Size, std::size_t elem1Size,
         std::size_t pool2Size, std::size_t elem2Size,
         std::size_t pool3Size, std::size_t elem3Size,
         std::size_t pool4Size, std::size_t elem4Size>
void BlockAllocator<pool1Size, elem1Size, pool2Size, elem2Size,
                    pool3Size, elem3Size, pool4Size, elem4Size>::
                         deallocate(void*& ptr) noexcept
{
  if (ptr == nullptr)
    return;

  if (pool1Addr <= ptr && ptr < pool1Addr + pool1Size)
    pool1.deallocate(ptr);
  else if (pool2Addr <= ptr && ptr < pool2Addr + pool2Size)
    pool2.deallocate(ptr);
  else if (pool3Addr <= ptr && ptr < pool3Addr + pool3Size)
    pool3.deallocate(ptr);
  else if (pool4Addr <= ptr && ptr < pool4Addr + pool4Size)
    pool4.deallocate(ptr);
}

} /* namespace dynamicMemory */
#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 An-wendung spart man sich dadurch den Formatstring (hier ist g++ smart).

```
#include <cstdio>
int main(void)
{
  const char* pc1 = "Hello World";
  const char* const pc2 = "World";
  printf("%s\n", pc1);
  printf("%s\n", pc2);
  return 0;
}
```

```
        .file   "main.cpp"
        .text
        .section        .rodata
.LC0:
        .string "Hello World"
.LC1:
        .string "World"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
```

```
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        leaq    .LC0(%rip), %rax
        movq    %rax, -16(%rbp)
        leaq    .LC1(%rip), %rax
        movq    %rax, -8(%rbp)
        movq    -16(%rbp), %rax
        movq    %rax, %rdi
        call    puts@PLT
        leaq    .LC1(%rip), %rdi
        call    puts@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .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 Ta-bellen 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;`

```
#include <cstdio>
int main(void)
{
  int table0[] = {1, 2, 3};
  const int table1[] = {1, 2, 3};
  static const int table2[] = {1, 2, 3};
  static int table3[] = {1, 2, 3};

  for(int i=0; i<3; ++i)
  {
    printf("%d\n", table0[i]);
    printf("%d\n", table1[2-i]);
    printf("%d\n", table2[i]+2);
    printf("%d\n", table3[2-i]);
  }
  return 0;
}
```

```
        .file   "main.cpp"
        .text
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "%d\n"
        .section        .text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB30:
        .cfi_startproc
        pushq   %r15
        .cfi_def_cfa_offset 16
        .cfi_offset 15, -16
        pushq   %r14
        .cfi_def_cfa_offset 24
        .cfi_offset 14, -24
        leaq    _ZZ4mainE6table2(%rip), %r14
        pushq   %r13
        .cfi_def_cfa_offset 32
        .cfi_offset 13, -32
        pushq   %r12
        .cfi_def_cfa_offset 40
        .cfi_offset 12, -40
        leaq    _ZZ4mainE6table3(%rip), %r13
        pushq   %rbp
        .cfi_def_cfa_offset 48
        .cfi_offset 6, -48
        pushq   %rbx
        .cfi_def_cfa_offset 56
        .cfi_offset 3, -56
        leaq    .LC0(%rip), %rbp
        movl    $8, %ebx
        xorl    %r12d, %r12d
        subq    $56, %rsp
        .cfi_def_cfa_offset 112
        movq    %fs:40, %rax
        movq    %rax, 40(%rsp)
        xorl    %eax, %eax
        leaq    16(%rsp), %rcx
        movabsq $8589934593, %rax
        leaq    28(%rsp), %r15
        movq    %rax, 16(%rsp)
        movl    $3, 24(%rsp)
        movq    %rax, 28(%rsp)
        movl    $3, 36(%rsp)
.L2:
        movl    (%rcx,%r12), %edx
        movq    %rbp, %rsi
        movl    $1, %edi
        xorl    %eax, %eax
        movq    %rcx, 8(%rsp)
        call    __printf_chk@PLT
        movl    (%r15,%rbx), %edx
        movq    %rbp, %rsi
        movl    $1, %edi
        xorl    %eax, %eax
        call    __printf_chk@PLT
        movl    (%r14,%r12), %eax
        movq    %rbp, %rsi
        movl    $1, %edi
        addq    $4, %r12
        leal    2(%rax), %edx
        xorl    %eax, %eax
        call    __printf_chk@PLT
        movl    0(%r13,%rbx), %edx
        xorl    %eax, %eax
        movq    %rbp, %rsi
        movl    $1, %edi
        subq    $4, %rbx
        call    __printf_chk@PLT
        cmpq    $-4, %rbx
        movq    8(%rsp), %rcx
        jne     .L2
        xorl    %eax, %eax
        movq    40(%rsp), %rcx
        xorq    %fs:40, %rcx
        jne     .L7
        addq    $56, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 56
        popq    %rbx
        .cfi_def_cfa_offset 48
        popq    %rbp
        .cfi_def_cfa_offset 40
        popq    %r12
        .cfi_def_cfa_offset 32
```

```
        popq    %r13
        .cfi_def_cfa_offset 24
        popq    %r14
        .cfi_def_cfa_offset 16
        popq    %r15
        .cfi_def_cfa_offset 8
        ret
.L7:
        .cfi_restore_state
        call    __stack_chk_fail@PLT
        .cfi_endproc
.LFE30:
        .size   main, .-main
        .section        .rodata
        .align 8
        .type   _ZZ4mainE6table2, @object
        .size   _ZZ4mainE6table2, 12
_ZZ4mainE6table2:
        .long   1
        .long   2
        .long   3
        .set    _ZZ4mainE6table3,_ZZ4mainE6table2
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .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 wer-den 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.

```
#include <cstdio>
#include "ints.h"

const int const1 = 64;
const int const2 = 128;
const int const3 = 256;

int main(void)
{
  int tmp1;
  int tmp2 = 20;
  tmp1 = MAKRO_DEF1;
  tmp2 = tmp2 + MAKRO_DEF1;
  printf("Mit Makros: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);

  tmp2 = 20;
  tmp1 = en1;
  tmp2 = tmp2 + en1;
  printf("Mit Enums: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);

  tmp2 = 20;
  tmp1 = const3;
  tmp2 = tmp2 + const3;
  printf("Mit Const: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);

  tmp2 = foo(tmp1);
  printf("Unterprogramm: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);

  return 0;
}
```

```
        .file   "main.cpp"
        .text
        .globl  const1
        .section        .rodata
        .align 4
        .type   const1, @object
        .size   const1, 4
const1:
        .long   64
        .globl  const2
        .align 4
        .type   const2, @object
        .size   const2, 4
const2:
        .long   128
        .globl  const3
        .align 4
        .type   const3, @object
        .size   const3, 4
const3:
        .long   256
        .align 8
.LC0:
        .string "Mit Makros: tmp1 = %d, tmp2 = %d\n"
        .align 8
.LC1:
        .string "Mit Enums: tmp1 = %d, tmp2 = %d\n"
        .align 8
.LC2:
        .string "Mit Const: tmp1 = %d, tmp2 = %d\n"
        .align 8
.LC3:
        .string "Unterprogramm: tmp1 = %d, tmp2 = %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $20, -8(%rbp)
        movl    $1, -4(%rbp)
        addl    $1, -8(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $20, -8(%rbp)
        movl    $8, -4(%rbp)
        addl    $8, -8(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC1(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $20, -8(%rbp)
        movl    $256, -4(%rbp)
        addl    $256, -8(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC2(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    -4(%rbp), %eax
        movl    %eax, %edi
        call    _Z3fooi@PLT
        movl    %eax, -8(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
```

```
        movl    %eax, %esi
        leaq    .LC3(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

```
#define MAKRO_DEF1 1
#define MAKRO_DEF2 2
#define MAKRO_DEF3 4

enum
{
  en1 = 8,
  en2 = 16,
  en3 = 32
};

extern const int const1;
extern const int const2;
extern const int const3;

int foo(int arg);
```

```
#include "ints.h"

int foo(int arg)
{
  return arg + MAKRO_DEF1 + en1 + const3;
}
```

```
        .file   "subs.cpp"
        .text
        .globl  _Z3fooi
        .type   _Z3fooi, @function
_Z3fooi:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    %edi, -4(%rbp)
        movl    -4(%rbp), %eax
        leal    9(%rax), %edx
        movl    const3(%rip), %eax
        addl    %edx, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   _Z3fooi, .-_Z3fooi
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

4. siehe ./Loesung/A1-3 Gleitpunktkonstanten können normalerweise (speziell bei Mikrocontrollern) nicht mittels Immediate-Ad-ressierung 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 MOV-ABSQ (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 densel-ben 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

```
#include <cstdio>
#include "floats.h"

const double const1 = 432.22128;
const double const2 = 23.55128;
const double const3 = 256.56;

int main(void)
{
  double tmp1;
  double tmp2 = 2.5;
  tmp1 = MAKRO_DEF1;
  tmp2 = tmp2 + MAKRO_DEF1;
  printf("Mit Makros: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2);

  tmp2 = 2.5;
  tmp1 = const3;
  tmp2 = tmp2 + const3;
  printf("Mit Const: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2);

  tmp2 = foo(tmp1);
  printf("Unterprogramm: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2);

  return 0;
}
```

```
        .file   "main.cpp"
        .text
        .globl  const1
        .section        .rodata
        .align 8
        .type   const1, @object
        .size   const1, 8
const1:
        .long   1558557732
        .long   1081803658
        .globl  const2
        .align 8
        .type   const2, @object
        .size   const2, 8
const2:
        .long   2946691162
        .long   1077382432
        .globl  const3
        .align 8
        .type   const3, @object
        .size   const3, 8
const3:
        .long   3264175145
        .long   1081084149
        .align 8
.LC2:
        .string "Mit Makros: tmp1 = %f, tmp2 = %f\n"
        .align 8
.LC4:
        .string "Mit Const: tmp1 = %f, tmp2 = %f\n"
        .align 8
.LC5:
        .string "Unterprogramm: tmp1 = %f, tmp2 = %f\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movsd   .LC0(%rip), %xmm0
        movsd   %xmm0, -16(%rbp)
        movsd   .LC1(%rip), %xmm0
        movsd   %xmm0, -8(%rbp)
        movsd   -16(%rbp), %xmm1
        movsd   .LC1(%rip), %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -16(%rbp)
        movsd   -16(%rbp), %xmm0
        movq    -8(%rbp), %rax
        movapd  %xmm0, %xmm1
        movq    %rax, -24(%rbp)
        movsd   -24(%rbp), %xmm0
        leaq    .LC2(%rip), %rdi
        movl    $2, %eax
        call    printf@PLT
        movsd   .LC0(%rip), %xmm0
        movsd   %xmm0, -16(%rbp)
        movsd   .LC3(%rip), %xmm0
        movsd   %xmm0, -8(%rbp)
        movsd   -16(%rbp), %xmm1
        movsd   .LC3(%rip), %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -16(%rbp)
        movsd   -16(%rbp), %xmm0
        movq    -8(%rbp), %rax
        movapd  %xmm0, %xmm1
        movq    %rax, -24(%rbp)
        movsd   -24(%rbp), %xmm0
        leaq    .LC4(%rip), %rdi
        movl    $2, %eax
        call    printf@PLT
        movq    -8(%rbp), %rax
        movq    %rax, -24(%rbp)
        movsd   -24(%rbp), %xmm0
        call    _Z3food@PLT
        movq    %xmm0, %rax
        movq    %rax, -16(%rbp)
        movsd   -16(%rbp), %xmm0
        movq    -8(%rbp), %rax
        movapd  %xmm0, %xmm1
        movq    %rax, -24(%rbp)
        movsd   -24(%rbp), %xmm0
        leaq    .LC5(%rip), %rdi
        movl    $2, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .rodata
        .align 8
.LC0:
        .long   0
        .long   1074003968
        .align 8
.LC1:
        .long   1271310320
        .long   1074039095
        .align 8
.LC3:
        .long   3264175145
        .long   1081084149
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

```
#define MAKRO_DEF1 2.567
#define MAKRO_DEF2 234.6532
#define MAKRO_DEF3 4345.634

extern const double const1;
extern const double const2;
extern const double const3;

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 GIO 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 GIO Peripherie aufgelistet.
3. Verifizieren Sie die Funktionalität der Applik

11.2.1 Lösung

1. just do it
2.

```c
/*
 * main.c
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#include <stdint.h>


#define  EALLOW __asm(" EALLOW")
#define  EDIS   __asm(" EDIS")


// portA
#define GP_A_CTRL (*((volatile uint32_t *)((uint32_t)0x00006F80)))
#define GP_A_SEL1 (*((volatile uint32_t *)((uint32_t)0x00006F82)))
#define GP_A_SEL2 (*((volatile uint32_t *)((uint32_t)0x00006F84)))
#define GP_A_MUX1 (*((volatile uint32_t *)((uint32_t)0x00006F86)))
#define GP_A_MUX2 (*((volatile uint32_t *)((uint32_t)0x00006F88)))
#define GP_A_DIR  (*((volatile uint32_t *)((uint16_t)0x00006F8A)))
#define GP_A_PUD  (*((volatile uint32_t *)((uint32_t)0x00006F8C)))

#define GP_A_DATA (*((volatile uint32_t *)((uint32_t)0x00006FC0)))
#define GP_A_SET  (*((volatile uint32_t *)((uint32_t)0x00006FC2)))
#define GP_A_CLR  (*((volatile uint32_t *)((uint32_t)0x00006FC4)))
#define GP_A_TOG  (*((volatile uint32_t *)((uint32_t)0x00006FC6)))


// bit manipulation
#define SET_BIT(reg, bit) ((reg) |=  (((uint32_t)1u) << ((uint32_t)(bit))))
#define CLR_BIT(reg, bit) ((reg) &= ~(((uint32_t)1u) << ((uint32_t)(bit))))
#define GET_BIT(reg, bit) (((reg) & (((uint32_t)1u) << ((uint32_t)(bit)))) >> ((uint32_t)(bit)))

#define SET_BITS(reg, bits)     ((reg) |=  (bits))
#define CLR_BITS(reg, bits)     ((reg) &= ~(bits))
#define ARE_BITS_SET(reg, bits) (((reg) & (bits)) == (bits))


// pin configuration
#define LED1_PIN     6u
#define LED2_PIN     2u
#define SWITCH1_PIN  26u
#define SWITCH2_PIN  30u

int main(void)
{
  EALLOW; // allow register changes

  // init GIOs
  GP_A_CTRL = (uint32_t)0xFFFFFFFF; // control register
  GP_A_SEL1 = (uint32_t)0; // Sync for Pin 0 - Pin 15
  GP_A_SEL2 = (uint32_t)0; // Sync for Pin 16 - Pin 31
  GP_A_MUX1 = (uint32_t)0; // GIO for Pin 0 - Pin 15
  GP_A_MUX2 = (uint32_t)0; // GIO for Pin 16 - Pin 31
  GP_A_DIR =  (uint32_t)0; // all GIO's as input
  GP_A_PUD =  (uint32_t)0xFFFFFFFF; // disable GIO pullup for all
       GIOs

  // LED 1 - statusIndicator
  SET_BIT(GP_A_DIR, LED1_PIN); // configure direction output

  // LED 2 - errorIndicator
  SET_BIT(GP_A_DIR, LED2_PIN); // configure direction output

  // SWITCH 1 - statusSwitch
  CLR_BIT(GP_A_DIR, SWITCH1_PIN); // configure direction input

  // SWITCH 2 - errorSwitch
  CLR_BIT(GP_A_DIR, SWITCH2_PIN); // configure direction input

  EDIS; // disallow register changes

  while(1)
  {
    if(!ARE_BITS_SET(GP_A_DATA, 1ul << SWITCH1_PIN))
      SET_BITS(GP_A_SET,  1ul << LED1_PIN);
    else
      SET_BITS(GP_A_CLR,  1ul << LED1_PIN);

    if(!ARE_BITS_SET(GP_A_DATA,  1ul << SWITCH2_PIN))
      SET_BITS(GP_A_SET,  1ul << LED2_PIN);
    else
      SET_BITS(GP_A_CLR,  1ul << LED2_PIN);
  }

  return 0;
}
```

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.
```c
/*
 * main.c
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#include <bsl/include/led.h>
#include <bsl/include/switch.h>

int main(void)
{
  bsl_Led statusIndicator;
  bsl_ledInit(&statusIndicator, bsl_led1);

  bsl_Led errorIndicator;
  bsl_ledInit(&errorIndicator, bsl_led2);

  bsl_Switch statusSwitch;
  bsl_switchInit(&statusSwitch, bsl_switch1);

  bsl_Switch errorSwitch;
  bsl_switchInit(&errorSwitch, bsl_switch2);

```

```
  while(1)
  {
    if(bsl_switchPressed(&statusSwitch))
      bsl_ledOn(&statusIndicator);
    else
      bsl_ledOff(&statusIndicator);

    if(bsl_switchPressed(&errorSwitch))
      bsl_ledOn(&errorIndicator);
    else
      bsl_ledOff(&errorIndicator);
  }

  return 0;
}
```

```
/*
 * led.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef BSL_LED_H_
#define BSL_LED_H_

#include <csl/include/pin.h>

/**
 * @brief The EswRobot LED ID.
 */
typedef enum
{
  bsl_led1,
  bsl_led2
} bsl_LedId;

/**
 * @brief The EswRobot LED data.
 */
typedef struct
{
  csl_Pin pin; ///< GIO pin
} bsl_Led;

/**
 * @brief Initializes the EswRobot LED.
 *
 * @param led LED data
 * @param id LED ID
 *
 * @pre @p led pointer must be valid, validity is not checked!
 */
static inline void bsl_ledInit(bsl_Led* led, bsl_LedId id)
{
  csl_pinInit(&(led->pin), (bsl_led1 == id) ? csl_pin6 : csl_pin2, csl_pinFun1);
  csl_pinSetDirection(&(led->pin), csl_pinOut);
}

/**
 * @brief Turns off the EswRobot LED.
 *
 * @param led LED data
 *
 * @pre @p led pointer must be valid, validity is not checked!
 */
static inline void bsl_ledOff(bsl_Led* led)
{
  csl_pinClear(&(led->pin));
}

/**
 * @brief Turns on the EswRobot LED.
 *
 * @param led LED data
 *
 * @pre @p led pointer must be valid, validity is not checked!
 */
static inline void bsl_ledOn(bsl_Led* led)
{
  csl_pinSet(&(led->pin));
}

#endif /* BSL_LED_H_ */
```

```
/*
 * switch.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef BSL_SWITCH_H_
#define BSL_SWITCH_H_

#include <csl/include/pin.h>

/**
 * @brief The EswRobot switch ID.
 */
typedef enum
{
  bsl_switch1,
  bsl_switch2
} bsl_SwitchId;

/**
 * @brief The EswRobot switch data.
 */
typedef struct
{
  csl_Pin pin;
} bsl_Switch;

/**
 * @brief Initializes the EswRobot switch.
 *
 * @param sw switch data
 * @param id switch ID
 *
 * @pre @p sw pointer must be valid, validity is not checked!
 */
static inline void bsl_switchInit(bsl_Switch* sw, bsl_SwitchId id)
{
  csl_pinInit(&(sw->pin), (bsl_switch1 == id) ? csl_pin26 : csl_pin30, csl_pinFun1);
  csl_pinSetDirection(&(sw->pin), csl_pinIn);
}

/**
 * @brief Returns if the EswRobot switch is pressed.
 *
 * @param sw switch data
 *
 * @pre @p sw pointer must be valid, validity is not checked!
 */
static inline bool bsl_switchPressed(const bsl_Switch* sw)
{
  return !csl_pinGet(&(sw->pin));
}

#endif /* BSL_SWITCH_H_ */
```

```
/*
 * bits.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_BITS_H_
#define CSL_BITS_H_

/**
 * @brief Bits
 */
enum
{
  csl_bit0  = 0x00000001, csl_bit1  = 0x00000002, csl_bit2  = 0x00000004, csl_bit3  = 0x00000008,
  csl_bit4  = 0x00000010, csl_bit5  = 0x00000020, csl_bit6  = 0x00000040, csl_bit7  = 0x00000080,
  csl_bit8  = 0x00000100, csl_bit9  = 0x00000200, csl_bit10 = 0x00000400, csl_bit11 = 0x00000800,
  csl_bit12 = 0x00001000, csl_bit13 = 0x00002000, csl_bit14 = 0x00004000, csl_bit15 = 0x00008000,
  csl_bit16 = 0x00010000, csl_bit17 = 0x00020000, csl_bit18 = 0x00040000, csl_bit19 = 0x00080000,
  csl_bit20 = 0x00100000, csl_bit21 = 0x00200000, csl_bit22 = 0
```

```
      x00400000, csl_bit23 = 0x00800000,
  csl_bit24 = 0x01000000, csl_bit25 = 0x02000000, csl_bit26 = 0
      x04000000, csl_bit27 = 0x08000000,
  csl_bit28 = 0x10000000, csl_bit29 = 0x20000000, csl_bit30 = 0
      x40000000, csl_bit31 = 0x80000000
};

#endif /* CSL_BITS_H_ */
```

```
/*
 * hwReg.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_HWREG_H_
#define CSL_HWREG_H_

#include <stdint.h>
#include <stdbool.h>

#include "./bits.h"

/**
 * @brief Sets the specified bits in the 32 bit register.
 *
 * @param reg register
 * @param bits bits to set
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg32SetBits(volatile uint32_t* reg,
    uint32_t bits)
{
  *reg |= bits;
}

/**
 * @brief Clears the specified bits in the 32 bit register.
 *
 * @param reg register
 * @param bits bits to clear
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg32ClearBits(volatile uint32_t* reg,
    uint32_t bits)
{
  *reg &= ~bits;
}

/**
 * @brief Returns if the bits are set in the 32 bit register.
 *
 * @param reg register
 * @param bits bits to check
 * @return true if the bits are set and false otherwise
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline bool csl_hwReg32AreBitsSet(const volatile uint32_t* reg
    , uint32_t bits)
{
  return (*reg & bits) == bits;
}

/**
 * @brief Sets the value in the 32 bit register.
 *
 * @param reg register
 * @param value value to set
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg32SetValue(volatile uint32_t* reg,
    uint32_t value)
{
  *reg = value;
}

/**
 * @brief Sets the 2 bit sub-value in the 32 bit register.
 *
 * @param reg register
 * @param value sub-value to set
 * @param shift shift for the sub-value
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg32Set2BitSubValue(volatile uint32_t* reg,
    uint32_t value, unsigned int shift)
{
  csl_hwReg32ClearBits(reg, (uint32_t)((1 << 2) - 1) << shift);
  csl_hwReg32SetBits(reg, value << shift);
}

/**
 * @brief Sets the specified bits in the 16 bit register.
 *
 * @param reg register
 * @param bits bits to set
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg16SetBits(volatile uint16_t* reg,
    uint16_t bits)
{
  *reg |= bits;
}

/**
 * @brief Clears the specified bits in the 16 bit register.
 *
 * @param reg register
 * @param bits bits to clear
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg16ClearBits(volatile uint16_t* reg,
    uint16_t bits)
{
  *reg &= ~bits;
}

/**
 * @brief Returns if the bits are set in the 16 bit register.
 *
 * @param reg register
 * @param bits bits to check
 * @return true if the bits are set and false otherwise
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline bool csl_hwReg16AreBitsSet(const volatile uint16_t* reg
    , uint16_t bits)
{
  return (*reg & bits) == bits;
}

/**
 * @brief Sets the value in the 16 bit register.
 *
 * @param reg register
 * @param value value to set
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg16SetValue(volatile uint16_t* reg,
    uint16_t value)
{
  *reg = value;
}

/**
 * @brief Sets the 2 bit sub-value in the 16 bit register.
 *
 * @param reg register
 * @param value sub-value to set
 * @param shift shift for the sub-value
 *
 * @pre @p reg pointer must be valid, validity is not checked!
 */
static inline void csl_hwReg16Set2BitSubValue(volatile uint16_t* reg,
    uint16_t value, unsigned int shift)
{
  csl_hwReg16ClearBits(reg, (uint16_t)((1 << 2) - 1) << shift);
  csl_hwReg16SetBits(reg, value << shift);
}

#endif /* CSL_HWREG_H_ */
```

```c
/*
 * pin.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_PIN_H_
#define CSL_PIN_H_

#include "./bits.h"
#include "./hwReg.h"
#include "./port.h"

/**
 * @brief The C2000 pin ID.
 */
typedef enum
{
  csl_pin0 = 0, // must be 0
  csl_pin1,
  csl_pin2,
  csl_pin3,
  csl_pin4,
  csl_pin5,
  csl_pin6,
  csl_pin7,
  csl_pin8,
  csl_pin9,
  csl_pin10,
  csl_pin11,
  csl_pin12,
  csl_pin13,
  csl_pin14,
  csl_pin15,
  csl_pin16,
  csl_pin17,
  csl_pin18,
  csl_pin19,
  csl_pin20,
  csl_pin21,
  csl_pin22,
  csl_pin23,
  csl_pin24,
  csl_pin25,
  csl_pin26,
  csl_pin27,
  csl_pin28,
  csl_pin29,
  csl_pin30,
  csl_pin31,
  csl_pin32,
  csl_pin33,
  csl_pin34,
  csl_pin35,
  csl_pin36,
  csl_pin37,
  csl_pin38,
  csl_pin39,
  csl_pin40,
  csl_pin41,
  csl_pin42,
  csl_pin43,
  csl_pin44,
  csl_pin45,
  csl_pin46,
  csl_pin47,
  csl_pin48,
  csl_pin49,
  csl_pin50,
  csl_pin51,
  csl_pin52,
  csl_pin53,
  csl_pin54,
  csl_pin55,
  csl_pin56,
  csl_pin57,
  csl_pin58,
  csl_pin59,
  csl_pin60,
  csl_pin61,
  csl_pin62,
  csl_pin63
} csl_PinId;

/**
 * @brief The pin data.
 */
typedef struct
{
  csl_Port port;
  uint32_t bit;
} csl_Pin;

/**
 * @brief The pin direction.
 */
typedef enum
{
  csl_pinIn,
  csl_pinOut
} csl_PinDirection;

/**
 * @brief The pin mux.
 */
typedef enum
{
  csl_pinFun1 = 0, ///< GIO
  csl_pinFun2 = csl_bit0,
  csl_pinFun3 = csl_bit1,
  csl_pinFun4 = csl_bit1 | csl_bit0
} csl_PinMux;

/**
 * @brief Initializes the C2000 pin to the default configuration.
 *
 * @param pin pin data
 * @param id pin ID
 * @param mux pin mux
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
void csl_pinInit(csl_Pin* pin, csl_PinId id, csl_PinMux mux);

/**
 * @brief Sets the direction of the C2000 pin.
 *
 * @param pin pin data
 * @param dir direction
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
void csl_pinSetDirection(csl_Pin* pin, csl_PinDirection dir);

/**
 * @brief Sets the the C2000 pin.
 *
 * @param pin pin data
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
static inline void csl_pinSet(csl_Pin* pin)
{
  csl_hwReg32SetBits(&(pin->port.data->set), pin->bit);
}

/**
 * @brief Clears the C2000 pin.

 * @param pin pin data
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
static inline void csl_pinClear(csl_Pin* pin)
{
  csl_hwReg32SetBits(&(pin->port.data->clear), pin->bit);
}

/**
 * @brief Toggles the C2000 pin.

 * @param pin pin data
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
static inline void csl_pinToggle(csl_Pin* pin)
{
  csl_hwReg32SetBits(&(pin->port.data->toggle), pin->bit);
}

/**
 * @brief Returns the state of the C2000 pin.
 *
 * @param pin pin data
 * @return pin state
 *
 * @pre @p pin pointer must be valid, validity is not checked!
 */
static inline bool csl_pinGet(const csl_Pin* pin)
{
  return csl_hwReg32AreBitsSet(&(pin->port.data->dat), pin->bit);
}

#endif /* CSL_PIN_H_ */
```

```c
/*
 * port.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_PORT_H_

#define CSL_PORT_H_

#include <stdint.h>
#include <stdbool.h>

/**
 * @brief The C2000 port control register.
 */
```

```
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 period.
57   *
58   * @param port port data
59   * @param id port ID
60   * @param initSamplePeriod initializes the sample period if set
61   *
62   * @note This function is called automatically by \ref csl_pinInit.
63   * @pre @p port pointer must be valid, validity is not checked!
64   */
65  void csl_portInit(csl_Port* port, csl_PortId id, bool
        initSamplePeriod);
66
67  #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 -
          numPinsPerPort);
46
47    // sync
48    bool highReg = bitNr >= (numPinsPerPort/syncBitWidth);
49    unsigned int shift = (highReg ? bitNr - (numPinsPerPort/
          syncBitWidth) : bitNr) * syncBitWidth;
50    csl_hwReg32Set2BitSubValue(&(pin->port.ctrl->sel[highReg]),
          syncToSysClk, shift);
51
52    // mux
53    highReg = bitNr >= (numPinsPerPort/muxBitWidth);
54    shift = (highReg ? bitNr - (numPinsPerPort/muxBitWidth) : bitNr) *
          muxBitWidth;
55    csl_hwReg32Set2BitSubValue(&(pin->port.ctrl->mux[highReg]), mux,
          shift);
56
57    // disable pull-up
58    csl_hwReg32SetBits(&(pin->port.ctrl->pud), pin->bit);
59
60    protHwRegAccessDisallow();
61
62    // default set to in
63    csl_pinSetDirection(pin, csl_pinIn);
64  }
65
66  void csl_pinSetDirection(csl_Pin* pin, csl_PinDirection dir)
67  {
68    protHwRegAccessAllow();
69
70    if(dir == csl_pinIn)
71    {
72      csl_hwReg32ClearBits(&(pin->port.ctrl->dir), pin->bit);
73    }
74    else
75    {
76      csl_hwReg32SetBits(&(pin->port.ctrl->dir), pin->bit);
77    }
78
79    protHwRegAccessDisallow();
80  }
```

```
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 control
          register of port A
16    addrPortBCtrlRegs = 0x00006F90, ///< base address of the control
          register of port B
17
18    addrPortADataRegs = 0x00006FC0, ///< base address of the data
          register of port A
19    addrPortBDataRegs = 0x00006FC8  ///< base address of the data
          register of port B
20  };
21
22  void csl_portInit(csl_Port* port, csl_PortId id, bool
        initSamplePeriod)
23  {
24    if(csl_portA == id)
25    {
26      port->ctrl = (csl_PortCtrlRegs*)addrPortACtrlRegs;
27      port->data = (csl_PortDataRegs*)addrPortADataRegs;
28    }
29    else
30    {
31      port->ctrl = (csl_PortCtrlRegs*)addrPortBCtrlRegs;
32      port->data = (csl_PortDataRegs*)addrPortBDataRegs;
33    }
```

```
  if(initSamplePeriod)
  {
    protHwRegAccessAllow();

    // Sets the sample period for the entire port to 510 * T_sysclk.
    // This is done, because the sample period cannot be set for each GIO pin separately.
    // More information can be found in the user manual of the C2000 DSP.
    if(!csl_hwReg32AreBitsSet(&(port->ctrl->ctrl), ~((uint32_t)0)))
    {
      csl_hwReg32SetValue(&(port->ctrl->ctrl), ~((uint32_t)0)); // sample period = 510 * T_sysclk
    }

    protHwRegAccessDisallow();
  }
}
```

```
/*
 * protHwRegAccess.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_PROTHWREGACCESS_H_
#define CSL_PROTHWREGACCESS_H_

/**
 * @brief Allows write access to protected C2000 register.
 */
static inline void protHwRegAccessAllow(void)
{
  __asm(" EALLOW");
}

/**
 * @brief Disallows write access to protected C2000 register.
 */
static inline void protHwRegAccessDisallow(void)
{
  __asm(" EDIS");
}

#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 Projekte 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.
```
/*
 * Led.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef BSL_LED_H_
#define BSL_LED_H_

#include <csl/include/Pin.h>

namespace bsl
{

/**
 * @brief Abstracts EswRobot LEDs.
 */
class Led
{
  public:
    /**
     * @brief The LED ID.
     */
    enum Id
    {
      led1,
      led2
    };

    /**
     * @brief Initializes the LED.
     *
     * @param id LED ID
     */
    Led(Id id) :
      pin(led1 == id ? csl::Pin::pin6 : csl::Pin::pin2,
          csl::Pin::out)
    {

    }

    /**
     * @brief Turns off the LED.
     */
    void off()
```

```
    {
      pin.clear();
    }

    /**
     * @brief Turns on the LED.
     */
    void on()
    {
      pin.set();
    }

    /**
     * @brief Initializes the LED.
     *
     * @param id LED ID
     */
    LED(Id id) :
      pin(led1 == id ? csl::Pin::pin27 : csl::Pin::pin31,
          csl::Pin::out)
    {

    }

  private:
    csl::Pin pin; ///< GIO pin

};

} // namespace bsl

#endif /* BSL_LED_H_ */
```

```
/*
 * Switch.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef BSL_SWITCH_H_
#define BSL_SWITCH_H_

#include <csl/include/Pin.h>

namespace bsl
{

/**
 * @brief Abstracts EswRobot switches.
 */
class Switch
{
  public:
    /**
     * @brief The switch ID.
     */
    enum Id
    {
      switch1,
      switch2
    };

    /**
     * @brief Initializes the switch.
     *
     * @param id switch ID
     */
    Switch(Id id) :
      pin(switch1 == id ? csl::Pin::pin26 : csl::Pin::pin30,
          csl::Pin::in)
    {

    }

    /**
     * @brief Returns if the switch is pressed.
     *
     * @return true if the switch is pressed and false otherwise
     */
    bool pressed() const
    {
      return !pin.get();
    }

  private:
    csl::Pin pin; ///< GIO pin
};

} // namespace bsl

#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 Projekte 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

```cpp
/*
 * HwReg.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_HWREG_H_
#define CSL_HWREG_H_

extern "C"
{
#include <stdint.h> // C2000 does not define <cstdint>
}

#include "./bits.h"

namespace csl
{

/**
 * @brief Abstracts C2000 hardware registers.
 *
 * @tparam RegType register unsigned int type must match the integer width of the
 * used architecture
 */
template<typename RegType>
class HwReg
{
  public:
    /**
     * @brief Sets the specified bits in the register.
     *
     * @param bits bits to set
     */
    void setBits(const RegType& bits)
    {
      reg |= bits;
    }

    /**
     * @brief Clears the specified bits in the register.
     *
     * @param bits bits to clear
     */
    void clearBits(const RegType& bits)
    {
      reg &= ~bits;
    }

    /**
     * @brief Returns if the bits are set in the register.
     *
     * @param bits bits to check
     *
     * @return true if the bits are set and false otherwise
     */
    bool areBitsSet(const RegType& bits) const
    {
      return (reg & bits) == bits;
    }

    /**
     * @brief Sets the value in the register.
     *
     * @param value value to set
     */
    void setValue(const RegType& value)
    {
      reg = value;
    }

    /**
     * @brief Sets the sub-value in the register.
     *
     * @tparam bitWidth bit width of the sub-value
     * @tparam shift shift for the sub-value
     *
     * @param value sub-value to set
     */
    template<unsigned int bitWidth, unsigned int shift>
    void setSubValue(RegType value) // const RegType& results in an linking error
    {
      clearBits(((static_cast<RegType>(1) << bitWidth) - static_cast<RegType>(1)) << shift);
      setBits(value << shift);
    }

    /**
     * @brief Sets the sub-value in the register.
     *
     * @tparam bitWidth bit width of the sub-value
     *
     * @param value sub-value to set
     * @param shift shift for the sub-value
     */
    template<unsigned int bitWidth>
    void setSubValue(const RegType& value, unsigned int shift)
    {
      clearBits(((static_cast<RegType>(1) << bitWidth) - static_cast<RegType>(1)) << shift);
      setBits(value << shift);
    }

  private:
    /**
     * @brief The destructor.
     */
    ~HwReg(); // prevent destructing of directly mapped register

    /**
     * @brief The copy-constructor.
     */
    HwReg(const HwReg&); // prevent copying of directly mapped register

    /**
     * @brief The assignment operator.
     */
    HwReg& operator=(const HwReg&); // prevent copying of directly mapped register

    volatile RegType reg; ///< register which is directly mapped to the register address
};

} // namespace csl

#endif /* CSL_HWREG_H_ */
```

3. In diesem Fall macht ein reinterpret_cast< HwRegister¡uint32_t>*> mehr Sinn, da keine Konstruktoraufruf 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 Konstruktoraufruf auf null Instruktionen reduziert. Siehe ./Loesung/CPPHAL_advanced/csl/include/Pin.h

```
/*
 * Pin.h
 *
 *  Created on: April 15, 2019
 *      Author: Gian Danuser
 */

#ifndef CSL_PIN_H_
#define CSL_PIN_H_

extern "C"
{
#include <stdint.h> // C2000 does not define <cstdint>
}

#include "./bits.h"
#include "./ProtHwRegAccess.h"
#include "./HwReg.h"
#include "./Port.h"

namespace csl
{

namespace pin
{

/**
 * @brief The pin ID.
 */
enum Id
{
  pin0 = 0, // must be 0
  pin1,
  pin2,
  pin3,
  pin4,
  pin5,
  pin6,
  pin7,
  pin8,
  pin9,
  pin10,
  pin11,
  pin12,
  pin13,
  pin14,
  pin15,
  pin16,
  pin17,
  pin18,
  pin19,
  pin20,
  pin21,
  pin22,
  pin23,
  pin24,
  pin25,
  pin26,
  pin27,
  pin28,
  pin29,
  pin30,
  pin31,
  pin32,
  pin33,
  pin34,
  pin35,
  pin36,
  pin37,
  pin38,
  pin39,
  pin40,
  pin41,
  pin42,
  pin43,
  pin44,
  pin45,
  pin46,
  pin47,
  pin48,
  pin49,
  pin50,
  pin51,
  pin52,
  pin53,
  pin54,
  pin55,
  pin56,
  pin57,
  pin58,
  pin59,
  pin60,
  pin61,
  pin62,
  pin63
};

/**
 * @brief The pin direction.
 */
enum Direction
{
  in,
  out
};

/**
 * @brief The pin mux.
 */
enum Mux
{
  fun1 = 0, ///< GIO
  fun2 = bit0,
  fun3 = bit1,
  fun4 = bit1|bit0
};

/**
 * @brief The pin sync selection.
 */
enum Sync
{
  syncToSysClk = 0 ///< sync to system clock
};

/**
 * @brief The C2000 pin base class.
 *
 * @tparam id port ID
 */
template<port::Id id>
class PinBase
{
public:
    /**
     * @brief Initializes the port if required.
     *
     * @param bitNr bit number
     * @param dir pin direction
     * @param mux pin mux
     * @param isPullupEn true if pull-up enabled
     * @param sync sync to specified clock
     */
    PinBase(unsigned int bitNr,
            Direction dir,
            Mux mux,
            bool isPullupEn,
            Sync sync);

    /**
     * @brief Sets the direction of the pin.
     *
     * @param bitNr bit number
     * @param dir pin direction
     */
    void setDirection(unsigned int bitNr, Direction dir);

private:
    enum
    {
      muxBitWidth = 2,  ///< bit width per mux setting
      syncBitWidth = 2, ///< bit width per sync setting
      pudBitWidth = 1   ///< bit width per pud setting
    };

    static bool initDonePort; ///< port initialized flag
};

/**
 * @brief Template meta programming to get the port ID derived from
      the pin ID.
 *
 * @tparam pinId pin ID
 */
template<Id pinId>
struct GetPort
{
  static const port::Id id = pinId < port::numPins ? port::a : port::
       b;
};

/**
 * @brief Abstracts C2000 pins.
 *
 * @tparam id pin ID
 */
template<Id id>
class Pin : public PinBase<GetPort<id>::id>
{

```

```
  public:

    /**
     * @brief Initializes the pin to the default configuration.
     *
     * @param dir pin direction (default: in)
     * @param mux pin mux (default: GIO)
     * @param isPullupEn true if pull-up enabled (default: enabled)
     * @param sync sync to specified clock (default: sync to system clock)
     */
    Pin(Direction dir = in,
        Mux mux = fun1,
        bool isPullupEn = true,
        Sync sync = syncToSysClk);

    /**
     * @brief Sets the direction of the pin.
     *
     * @param dir pin direction
     */
    void setDirection(Direction dir);

    /**
     * @brief Sets the pin.
     */
    void set();

    /**
     * @brief Clears the pin.
     */
    void clear();

    /**
     * @brief Toggles the pin.
     */
    void toggle();

    /**
     * @brief Returns the state of the pin.
     *
     * @return pin state
     */
    bool get() const;
};


//
// template meta programming
//

/**
 * @brief Template meta programming to get the bit number derived from the pin ID.
 *
 * @tparam id pin ID
 */
template<Id id>
struct GetBitNr
{
  static const unsigned int nr = id < port::numPins ? id : id - port::numPins;
};


//
// implementation
//

template<port::Id id>
bool PinBase<id>::initDonePort = false;

template<port::Id id>
PinBase<id>::PinBase(unsigned int bitNr,
                     Direction dir,
                     Mux mux,
                     bool isPullupEn,
                     Sync sync)
{
  // Ensures that each port is initialized only once.
  if(!initDonePort)
  {
    port::Port<id> port(true);
    initDonePort = true;
  }

  ProtHwRegAccess::allow();

  //configure sync
  if(bitNr >= (port::numPins/syncBitWidth))
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::sel2)->setSubValue<syncBitWidth>(
        sync, bitNr - port::numPins/syncBitWidth);
  else
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::sel1)->setSubValue<syncBitWidth>(
        sync, bitNr);

  //configure mux
  if(bitNr >= (port::numPins/muxBitWidth))
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::mux2)->setSubValue<syncBitWidth>(
        mux, bitNr - port::numPins/muxBitWidth);
  else
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::mux1)->setSubValue<syncBitWidth>(
        mux, bitNr);

  //configure pull-up
  reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::pud)->setSubValue<pudBitWidth>(
      isPullupEn, bitNr);

  ProtHwRegAccess::disallow();

  setDirection(bitNr, dir);
}

template<port::Id id>
void PinBase<id>::setDirection(unsigned int bitNr, Direction dir)
{
  ProtHwRegAccess::allow();

  if(dir == in)
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::dir)->clearBits(
        static_cast<uint32_t>(1) << bitNr);
  else
    reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id>::dir)->setBits(
        static_cast<uint32_t>(1) << bitNr);

  ProtHwRegAccess::disallow();
}

template<Id id>
Pin<id>::Pin(Direction dir,
             Mux mux,
             bool isPullupEn,
             Sync sync) :
  PinBase<GetPort<id>::id>(GetBitNr<id>::nr, dir, mux, isPullupEn, sync)
{

}

template<Id id>
void Pin<id>::setDirection(Direction dir)
{
  PinBase<id>::setDirection(GetBitNr<id>::nr, dir);
}

template<Id id>
inline void Pin<id>::set()
{
  // reinterpret cast is used instead of placement new,
  // because C2000 compiler does not reduce the new operator to zero instructions
  reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<GetPort<id>::id>::set)->setBits(
      static_cast<uint32_t>(1) << GetBitNr<id>::nr);
}

template<Id id>
inline void Pin<id>::clear()
{
  // reinterpret cast is used instead of placement new,
  // because C2000 compiler does not reduce the new operator to zero instructions
  reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<GetPort<id>::id>::clr)->setBits(
      static_cast<uint32_t>(1) << GetBitNr<id>::nr);
}

template<Id id>
inline void Pin<id>::toggle()
{
  // reinterpret cast is used instead of placement new,
  // because C2000 compiler does not reduce the new operator to zero instructions
  reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<GetPort<id>::id>::toggle)->setBits(
      static_cast<uint32_t>(1) << GetBitNr<id>::nr);
}

template<Id id>
inline bool Pin<id>::get() const
{
  // reinterpret cast is used instead of placement new,
  // because C2000 compiler does not reduce the new operator to zero instructions
  return reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<GetPort<id>::id>::dat)->areBitsSet(
      static_cast<uint32_t>(1) << GetBitNr<id>::nr);
}

} // namespace pin

} // namespace csl

#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/cp

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

```
//
      ============================================================================
// Name        : Distributions.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) HSR R. Bonderer
// Description : Tests various integer random distributions
//
      ============================================================================


#include <iostream>
#include <iomanip>
#include <random>
#include "RandIntGen.h"

using namespace std;

int main()
{
  enum
  {
    minValue = 1, // minimal value for random numbers
    maxValue = 20, // maximal value for random numbers
    stdDev = 2, // normal distribution standard deviation
    mean = 11, // normal distribution mean
    wbAlpha = 2, // weibull alpha
    wbBeta = 3, // weibull beta
    iterations = 20000 // number of iterations
  };

  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;
  }
  cout << "Uniform distribution (std library)" << endl;
  std::uniform_int_distribution<int> uniformDist(minValue, maxValue);
  for (int i = 0; i < iterations; ++i)
  {
    r = uniformDist(rgStd);
    ++histogram[r - minValue];
  }
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    cout << histogram[i] << " ";
  }
  cout << endl;

  // normal distribution
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    histogram[i] = 0;
  }
  cout << "Normal distribution" << endl;
  for (int i = 0; i < iterations; i++)
  {
    r = rg.getNormal(mean, stdDev);
    if (r >= 0 && r <= maxValue-minValue)    // r could be beyond array
        limits
      ++histogram[r];
  }
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    cout << histogram[i] << " ";
  }
  cout << endl;

  // normal distribution using std library random generators
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    histogram[i] = 0;
  }
  cout << "Normal distribution (std library)" << endl;
  std::normal_distribution<double> normalDist(mean, stdDev);
  for (int i = 0; i < iterations; i++)
  {
    r = std::round(normalDist(rgStd));
    if (r >= 0 && r <= maxValue-minValue)    // r could be beyond array
        limits
      ++histogram[r];
  }
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    cout << histogram[i] << " ";
  }
  cout << endl;

  // Weibull distribution
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    histogram[i] = 0;
  }
  cout << "Weibull distribution" << endl;
  for (int i = 0; i < iterations; ++i)
  {
    r = rg.getWeibull(wbAlpha, wbBeta);
    if (r >= 0 && r <= maxValue-minValue)    // r could be beyond array
        limits
      ++histogram[r];
  }
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    cout << histogram[i] << " ";
```

```
  }
  cout << endl;

  // Weibull distribution using std library random generators
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    histogram[i] = 0;
  }
  cout << "Weibull distribution (std library)" << endl;
  std::weibull_distribution<double> weibullDist(wbAlpha, wbBeta);
  for (int i = 0; i < iterations; ++i)
  {
    r = std::round(weibullDist(rgStd));
    if (r >= 0 && r <= maxValue-minValue)    // r could be beyond array
        limits
      ++histogram[r];
  }
  for (int i = 0; i < maxValue - minValue + 1; ++i)
  {
    cout << histogram[i] << " ";
  }
  cout << endl;


  return 0;
}
```

```
/*
 * Randgen.h
 *
 *  Created on: 19.03.2015
 *      Author: rbondere
 */

#ifndef RANDINTGEN_H_
#define RANDINTGEN_H_

class RandIntGen
{
  public:
    RandIntGen(int seed = 0);

    // returns a uniformly distributed random int value
    int getUniform(int low,              // lowest random value
                   int high) const;      // highest random value

    // returns a normally distributed random int value
    int getNormal(double mean,           // mean (expected value)
                  double sdev) const;    // standard deviation

    // returns a Weibull distributed random int value
    int getWeibull(double alpha,
                   double beta) const;
};

#endif /* RANDINTGEN_H_ */
```

```
/*
 * Randgen.cpp
 *
 *  Created on: 19.03.2015
 *      Author: rbondere
 */
#include <cstdlib>
#include <cmath>
#include "RandIntGen.h"

using namespace std;

RandIntGen::RandIntGen(int seed)
{
  srand(seed);
}

int RandIntGen::getUniform(int low,
                           int high) const
{
  return static_cast<int> (static_cast<double> (rand()) / (RAND_MAX +
      1.0)
                           * (high - low + 1)) + low;
  // '(RAND_MAX + 1.0)' ist notwendig, damit 0 <= u < 1.0 (ohne 1.0)
  // '+ 1' ist notwendig, weil nach int konvertiert wird.
  // Vor Konversion gibt es dadurch bei [1,20] Zahlen im Bereich
      [0.0,20.0[
  // Nach Konversion werden nur die ganzzahligen Anteile verwendet
  // Das hinterste "+ low" muss ausserhalb des Typecasts sein,
      andernfalls
  // ergibt es ein Ueberhoehung bei 0, falls low < 0 (alle Zahlen in
  // [-0.9999, +0.9999] wuerden zu 0, d.h. doppelt so viele wie erlaubt)
}

int RandIntGen::getNormal(double mean,
                          double sdev) const
{
  double v1;
  double v2;
  double s;
  do
  {
    v1 = 2 * static_cast<double> (rand()) / RAND_MAX - 1.0;
    v2 = 2 * static_cast<double> (rand()) / RAND_MAX - 1.0;
    s = v1 * v1 + v2 * v2;
  } while (s >= 1.0 || s == 0.0);
  double temp = sqrt(-2.0 * log(s) / s);
  return lround(v1 * temp * sdev + mean);
  // bei truncation (ohne lround(), sondern mit (int)) wird der
      Zufallswert
  // bei 0 doppelt gewertet!!
}

int RandIntGen::getWeibull(double alpha,
                           double beta) const
{
  double u;
  double w;
  u = static_cast<double> (rand()) / (RAND_MAX + 1.0); // 0.0 <= u < 1.0
  w = beta * pow(-log(1.0 - u), 1 / alpha);
  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: EF5CDDD830BBCBF544FFEB18E3B28B4D19D65A5AE16F4618D742683
img4.jpg: SHA256: EF5CDDD830BBCBF544FFEB18E3B28B4D19D65A5AE16F4618D742683
img5.jpg: SHA256: 5E8EDB576C3CF31013ED0858E35C280BA5B667164DA2991C1AE2ECAF

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.

```
//
      ============================================================================
// Name        : CipherFreq.cpp
// Author      : Reto Bonderer
// Version     :
// Copyright   : (c) Reto Bonderer
// Description : Attacks based on the char frequencies
// you may use the following knowledge for your attack:
// (1) key is 3 lowercase letters (a-z)
// (2) key is repeated as long as needed and xored byte by byte
// (3) the plaintext contains mixed cases, digits, punctuation and spaces
// (4) the plaintext is in English, the 10 most common letters in the
      English
//     language are, in order of decreasing frequency: e, t, a, o, i, n,
      s, h, r, d, l.
//
      ============================================================================



#include <iostream>
#include <iomanip>
using namespace std;

static const char ciphertext[] =
// to long to show

struct CharFreq
{
  char ch;
  unsigned int count;
};

// counts frequencies of a ciphertext character for the (chrPos+1)th key
      character
void countFrequencies(const char* cText,        // ciphertext
                      unsigned int cNumber,     // cText size
                      unsigned int chrPos,      // position of key char
                           (0..2)
                      CharFreq* f,              // array of char counters
                      unsigned int charNumber); // number of array elems
                           in f

// returns the key character from the most frequent ciphertext character
      in f
char getKeyChar(const CharFreq* f,              // array of char counters
                unsigned int charNumber);       // number of array elems
                     in f
```

```
// prints the first 'number' chars of the decrypted text
void printText(char first, char second, char third, const char* cText,
     unsigned int number);


int main(void)
{
  char key[3];

  cout << "Here is the key:" << endl;
  for (unsigned int k = 0; k < 3; ++k) // keylength = 3
  {
    CharFreq chFreq[256] = {{0, 0}};  // stores the frequencies of the
         chars

    // count freqs of (k+1)th key character
    countFrequencies(ciphertext, sizeof(ciphertext), k, chFreq, sizeof(
         chFreq)/sizeof(chFreq[0]));
    key[k] = getKeyChar(chFreq, sizeof(chFreq)/sizeof(chFreq[0]));
    cout << key[k];
  }

  cout << endl << endl << "Here is the full text:" << endl;
  printText(key[0], key[1], key[2], ciphertext, sizeof(ciphertext));

  return 0;
}

void countFrequencies(const char* cText, unsigned int cNumber, unsigned
     int chrPos,
                      CharFreq* f, unsigned int charNumber)
{
  unsigned int top = 0;  // index of next char to store
  for (unsigned int i = 0; i < cNumber; i+=3) // 3: keylength
  {
    unsigned int j;
    for (j = 0; j <= top; ++j)
    {
      if (cText[i+chrPos] == f[j].ch) // found entry
      {
        ++(f[j].count);
        break;
      }
    }
    if (j > top) // didn't find entry
    {
      f[top].ch = cText[i+chrPos];
```

```
      f[top].count = 1;
      ++top;
    }
  }
}

char getKeyChar(const CharFreq* f, unsigned int charNumber)
{
  // in fact: the most frequent character is the space character, 2nd is 'e' in English
  unsigned int count = f[0].count;
  unsigned int pos = 0;

  for (unsigned int i = 1; i < charNumber; ++i)
  {
    if (f[i].count > count)
    {
      count = f[i].count;
      pos = i;
    }
  }
  return f[pos].ch ^ 0x20; // space is the most frequent
}

void printText(char first, char second, char third, const char* cText,
     unsigned int number)
{
  char* pch = const_cast<char*>(cText);
  for (unsigned int i = 0; i < number/3; ++i)
  {
    char ch1 = first ^ *pch++;
    char ch2 = second ^ *pch++;
    char ch3 = third ^ *pch++;

    cout << ch1 << ch2 << ch3;
  }
  cout << endl;
}
```

`ciphertextByte = plaintextByte` $\oplus$ `key` und
`plaintextByte = ciphertextByte` $\oplus$ `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)