Programmierrichtlinien für C++

Die im folgenden beschriebenen Programmierrichtlinien sind für das SE-I-Praktikum verbindlich und müssen eingehalten werden.

1 Dateien

Für jede zu kodierende Klasse werden 2 Dateien angelegt, die Header(.h)-Datei und die Code(.C)-Datei. Die Dateien tragen den Namen der Klasse erweitert um die jeweilige Endung (.h oder .C). Die Zeilen in beiden Dateien sind maximal 80 Zeichen lang. Als Beispiel für die Gestaltung von Header- und Code-Datei dienen Any-Class.h und AnyClass.C.

Aufbau der Header-Datei

- Die Header-Datei beginnt immer mit den beiden Statements:
  #ifndef _KLASSE_H
  #define _KLASSE_H
  und endet mit der Zeile:
  #endif

  Dabei ist KLASSE durch den jeweiligen Klassennamen zu ersetzen.

- Im Kopf der Datei steht weiterhin (als Komentar) der Autor, das Erzeugungsdatum und die Dateihistorie. Die Historie ist eine Tabelle von Einträgen, bestehend aus: Datum, Autor, Gegenstand und Grund der Änderung.
- Danach kommen alle include-Statements, die benötigt werden.
- Unter Umständen folgt die Deklaration der Parametertypen.
- Darauf folgt die Klassendeklaration.
- Zu jeder Methode wird die Spezifikation als Kommentar angegeben.
- Inline-Code ist nicht erlaubt.

Aufbau der Code-Datei

- Im Kopf der Datei steht wie bei der Header-Datei (als Kommentar) der Autor, das Erzeugungsdatum und die Dateihistorie.
- Danach wird die Datei <Klasse.h> inkludiert:
  #include <Klasse.h>

  Dabei ist Klasse durch den jeweiligen Klassennamen zu ersetzen.

- Daran schließt sich der Quellcode für die einzelnen Methoden an.

2 Code

- Es müssen alle Methoden aus den Komponentenanforderungen und der Default-Konstruktor kodiert werden.
- In Klassen mit Bezügen auf andere Klassen muß ein expliziter Destruktor definiert sein oder ein Kommentar in der Klassendefinition stehen, warum der Default-Destruktor an dieser Stelle korrekt ist.
- Alle Methoden aus den Komponentenanforderungen, alle Konstruktoren und Destruktoren sind *public*. Ebenso sind die dem Timer übermittelten Funktionen *public*.
- Alle anderen Methoden und alle Attribute sind *private*.

- Globale Variablen sind nicht erlaubt.
- Variablen sollen nur am Anfang eines Blocks deklariert und initialisiert werden.
- In jeder Zeile steht nur eine Anweisung.
- Es sollen keine *type casts* verwendet werden. Ausnahme: Bei der Übergabe einer Funktion an den Timer muß ein *type cast* verwendet werden, um den Funktionsnamen an die Funktion TimerSet zu übergeben.
- Nur für Zähler sollen *for*-Schleifen verwendet werden. Für alles andere gibt es *while*.
- Einrückungen:

```
while (...) {

}
```

- Einrückungstiefe ist mindestens 2.
- Schleifenausgänge:
  - kein *continue*
  - *break* nur bei *switch*

3 Namenskonventionen

- Bezeichner tragen natürlichsprachliche, problemnahe Namen oder sind verständliche Abkürzungen solcher Namen.
- Schnittstellennamen werden von den Komponentenanforderungen übernommen.
- Zur optischen Trennung von Begriffen sollen nicht Underscores (_) sondern soll Groß- und Kleinschreibung verwendet werden.
- Kein Name beginnt mit einem Underscore (_).
- Schleifenvariablen werden durch einfache Kleinbuchstaben dargestellt.
- Variablennamen von Pointern enden auf Ptr.
- Klassennamen fangen mit Großbuchstaben an.

4 Kommentare *(aus W. Humphrey, „A Discipline For Software Engineering“)*

- Dokumetiere den Kode zum Verständnis eines Lesers
- Kommentare sollten Zweck und Verhalten erklären
- Variablendeklarationen sollten in einem Kommentar motiviert werden.

Beispiel:

guter Kommentar:

```
if (recordCount > limit) // have all records been processed
```

schlechter Kommentar:

```
if (recordCount > limit) // check if recordCount greater
```

- Kommentare düfen nur allein in einer Zeile oder am Ende einer Zeile stehen. Auf gar keinen Fall zwischen Anweisungen in einer Zeile. Gehen Kommentare über mehrere Zeilen, müssen am Anfang der Zeile Kommentarzeichen stehen.

5 Inhalt von Header- und Implementationsdatei

- Header- und eine Implementationsdateien sollten nach Möglichkeit jeweils nur eine Klasse enthalten.
- Die Dateien tragen den Namen der Klasse, erweitert um die jeweilige Endung (.h für Headerdatei und .C für Implementationsdatei). Die Dateinamen beginnen mit Kleinbuch-

staben.
- in einer Zeile sollen nicht mehr als 80 Zeichen stehen.
- In die Headerdatei werden diejenigen Files inkludiert, die in der Header- oder der Implementationsdatei benötigt werden. In der Implementationsdatei wird ausschließlich die Headerdatei der Klasse inkludiert.
- Deklaration und Definition von Attributen und Methoden geschehen in der Reihenfolge **private**, **protected** und **public** nach Attributen und Methoden getrennt.

Aufbau der Headerdatei

- Die Datei ist in eine *File section* und eine *Class section* getrennt. Die *File section* beschreibt Eigenschaften der Datei, während die *Class section* klassenspezifisch ist. Damit wird der Tatsache Rechnung getragen, daß mehrere Klassen innerhalb einer Datei deklariert werden können.
- An erster Stelle der *File section* stehen als Kommentar, der Name der Datei, ein Hinweis auf den Autoren, die Version und deren Zustand im Sinne der Produktverwaltung (z.B. experimental, stable, …) und auf das Erstellungsdatum. Eine Dateihistorie gibt an, wie sich die Datei im Laufe der Zeit verändert hat. Es handelt sich bei ihr um Einträge, bestehend aus: Datum, Autor, Gegenstand und Grund der Änderung.
- Anschließend werden die Headerdateien, die zur Compilierung des Header- und Implementierungsfiles der in der Datein deklarierten Klasse notwendig sind, inkludiert.
- Neben dem Namen der Klasse wird auch die Klasse allgemein beschrieben werden.
- Unter Umständen erfolgt eine Deklaration der Parametertypen.
- Danach werden die Klassen deklariert.

Beispiel: Headerdatei der Klasse AnyClass

```
//-------------------------------------------------------------------------------
//--------------------------------  File section  -------------------------------
//-------------------------------- Implementation -------------------------------
//-------------------------------------------------------------------------------
// Filename:       $RCSfile$
// Author:         $Author$
// State:          $State$
// Revision:       $Revision$
// Release:        $Name$
//
//----------------------------------- History -----------------------------------
//
// $Log$
//
//-------------------------------------------------------------------------------
//-------------------------------- Class section --------------------------------
//-------------------------------------------------------------------------------

#ifndef _FILENAME_H
#define _FILENAME_H

#include <AnyOtherClass.h>

//===============================================================================
// ClassName: AnyClass
//
// ClassDescription:
//      This class realizes the interface to the 'system time' and
//      manages the day interval for the attribute 'heating period'.
//===============================================================================

class AnyClass {
   private:
```

```
   // In this section all attributes and methods, which are
   // not visuable to other classes should be listed. The design
   // description must be added to each declaration of a method.

public:
     AnyClass(); // constructor
    ~AnyClass(); // destructor

   // In this section all methods, which are visuable to other
   // classes should be listed. The design description must be
   // added to each declaration of a method.

};              // class AnyClass

//================================================================================
// ClassName: AnyClass2
//
// ClassDescription:
// ...
//================================================================================
...
class AnyClass2 {
...
}               // Class AnyClass2

#endif // _FILENAME_H
```

Aufbau der Implementationsdatei

- Wie bei der Headerdatei ist die Datei in eine *File section* und eine *Class section* (s. o.) getrennt.
- Wie bei der Headerdatei stehen im Kopf der Datei Hinweise zum Autoren, zur Version und zum Datum. Im Gegensatz zu dieser wird aber auf die allgemeine Beschreibung der Klasse verzichtet, um Redundanzen zu vermeiden. Eine Dateihistorie wird, wie bei der Headerdatei, erstellt.
- Danach wird die Datei **Klasse.h** inkludiert
- #include <Klasse.h>. Dabei ist Klasse durch den jeweiligen Klassennamen zu ersetzen.
- Definition der Memberfunktionen in der Reihenfolge:
    - 1. Konstruktor(en)/Destruktor,
    - 2. dann alle anderen Memberfunktionen
- Die einzelnen Methodendefinitionen werden durch eine Kommentarzeile getrennt.

Beispiel: Implementationsdatei der Klasse AnyClass

```
//--------------------------------------------------------------------------------
//--------------------------------  File section  --------------------------------
//-------------------------------- Implementation --------------------------------
//--------------------------------------------------------------------------------
// Filename:       $RCSfile$
// Author:         $Author$
// State:          $State$
// Revision:       $Revision$
// Release:        $Name$
//
//----------------------------------- History -----------------------------------
//
// $Log$
//
//--------------------------------------------------------------------------------
//-------------------------------- Class section ---------------------------------
//--------------------------------------------------------------------------------

#include <AnyClass.h>
```

```
//==============================================================================
// ClassName: AnyClass
//==============================================================================

AnyClass:AnyClass()
{
…
} // AnyClass

//------------------------------------------------------------------------------
AnyClass:~AnyClass()
{
…
} // ~AnyClass
```