

Hon. Prof. Dr.-Ing. Hartmut Helmke
Ostfalia
Hochschule für angewandte
Wissenschaften
Fakultät für Informatik

Matrikelnummer:

Punktzahl:

Ergebnis:

Freiversuch F1 F2 F3

Klausur im WS 2020/21:

Die verschiedenen Programmierparadigmen von C++

Informatik Bachelorstudiengang Informatik Masterstudiengang

Es handelt sich hier um die Aufgaben für die Bachelor- Studenten.

Hilfsmittel wie Bücher und Skripte und eigene Notizen sind erlaubt.
Die Nutzung eines Computers z.B. mit einer Programmierumgebung oder einem Compiler ist gar nicht erlaubt!
Austausch von Hilfsmitteln mit Kommilitonen ist nicht erlaubt !
Die Kommunikation mit anderen Personen (zur Klausur) ist während der Klausur nicht erlaubt.

Bitte notieren Sie auf **allen** Blättern, die in die Bewertung eingehen sollen, Ihren Namen und Ihre Matrikelnummer.

Auf eine absolut korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte abgezogen.

Hinweis: In den folgenden Programmfragmenten wird die globale Variable `datei` verwendet. Hierfür kann der Einfachheit halber die Variable `cout` angenommen werden. Die Variable `datei` dient bei der Klausurerstellung lediglich dazu, automatisch eine Lösungsdatei zu erstellen.

Wir befinden uns jeweils im Namensraum `std`, d.h., ein `using namespace std;` dürfen Sie in jeder Codedatei annehmen. Außerdem dürfen Sie annehmen, dass für alle Code-Fragmente die erforderlichen `include`-Anweisungen für C++-Header-Dateien erfolgt sind.

Alle Lösungen sind auf **Extrablättern** mit vorangestellter Aufgabennummer zu notieren. Bitte vergessen Sie nicht, Ihren **Namen** auch auf den Extrablättern zu notieren.

Geplante Punktevergabe

Punktziel	Sonderpunkte	erreicht
Übungen:	XXX	
A1: 44 P.		
A2: 15 P.		
A3: 21 P.		
Summe 80+ SP.		

Aufgabe 1 : Minimale Standardschnittstelle

ca. 44 Punkte Das folgende Listing zeigt einen Ausschnitt aus der Klassendeklaration der Klasse `FileName`. Im Unterschied zur Vorlesung liegen auch Attribute im Heap-Speicher.

```
class FileName {
public:
    FileName(string name) {
        mp_min = nullptr;
        mp_sec = nullptr;
        SplitFilenameIntoDate (name);
    }
    FileName();
    ~FileName();
    void SplitFilenameIntoDate (string name);
    const int& GetMillisec () const {
        return m_millisec ;
    }
    const int& GetYear() const {
        return m_year;
    }
    const int& GetMonth() const {
        return m_month;
    }
    const int& GetDay() const {
        return m_day;
    }
    const string& GetGermanDate() const {
        return m_germanDate;
    }
    int GetSec() const { return *mp_sec; }
    void SetSec(int s) { *mp_sec =s; }
```

```
private:
    void SetDateGerman();
    int m_year;
    int m_month;
    int m_day;
    int m_hour;
    int* mp_min; // pointer to heap
    int* mp_sec; // pointer to heap
    int m_millisec ;
    string m_germanDate; // 20.1.2020
};
```

Die Implementierung einiger Methoden der Klasse zeigt das folgende Listing:

```
void FileName::SplitFilenameIntoDate (string astr_name)
/* 2019-02-15_11-32-40-00 is the filename, we split
into m_year, m_month ...
Same as in our lecture , but *mp_minutes and
*mp_seconds on heap
*/
{
    stringstream strStream (astr_name);
    char ch_dummy;
    strStream >> m_year;
    strStream >> ch_dummy;
    if (ch_dummy != '-') {
        return ;
    }
```

```
strStream >> m_month;
strStream >> ch_dummy;
if (ch_dummy != '-') {
    return;
}
strStream >> m_day;
char ch_dummy2;
strStream >> ch_dummy >> ch_dummy2;
if (ch_dummy != '_' || ch_dummy2 != '_') {
    return;
}

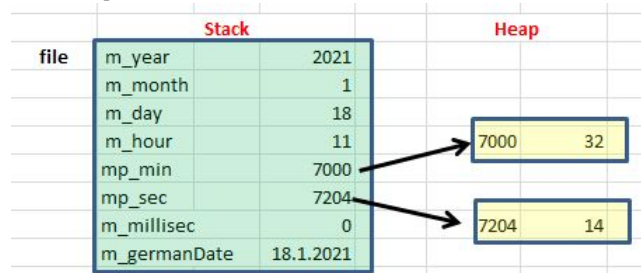
strStream >> m_hour;
strStream >> ch_dummy;
if (ch_dummy != '-') {
    return;
}
delete mp_min;
mp_min = new int; // This is important
strStream >> *mp_min;
strStream >> ch_dummy;
if (ch_dummy != '-') {
    return;
}
delete mp_sec;
mp_sec = new int; // This is important
strStream >> *mp_sec;
strStream >> ch_dummy;
if (ch_dummy != '-') {
    return;
}
strStream >> m_millisec ;
strStream >> ch_dummy;
if (ch_dummy != '-') {
    return;
}
SetDateGerman();
} // FileName:: SplitFilenameIntoDate ()

void FileName::SetDateGerman(){
    stringstream stream;
    stream << m_day << "."
        << m_month << "." << m_year;
    m_germanDate = stream.str();
}
FileName::~FileName() {
    delete mp_min;
    delete mp_sec;
}
```

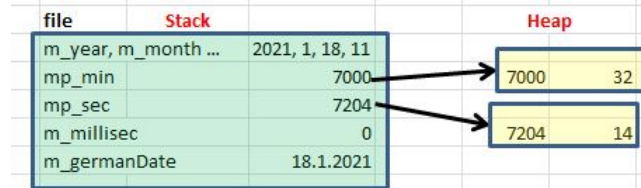
a.) (3 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `simple` ?

```
void simple() {
    FileName file ("2021-01-18__11-32-14-00");
    datei << "Year is " <<
        file .GetYear() << "\n"; /* I */
    datei << "Month is " <<
        file .GetMonth() << "\n";
    datei << "German Date " <<
        file .GetGermanDate() << "\n";
}
```

b.) (17 P.) Die folgende Zeichnung skizziert eine Speicherbelegung zum Zeitpunkt `/* 1 */` in der Funktion `simple`:



Sie dürfen und sollten in Ihren Skizzen vereinfachen zu:



Der Aufruf der folgenden Funktion `testBumm` führt zu undefiniertem Programmverhalten:

```
void changeToWrongSec(FileName val) {
    val.SetSec(72); /* 2 */
}
void testBumm() {
    FileName file("2021-01-18__11-32-14-00");
    datei << "Year is " <<
        file.GetYear() << "\n";
    changeToWrongSec(file); /* 3 */
    datei << "Year is now " <<
        file.GetYear() << "\n";
    datei << "Seconds are now " <<
        file.GetSec() << "\n";
}
```

Skizzieren Sie auf einem Extra-Blatt entsprechend der obigen Zeichnung die Speicherbelegungen zu den Zeitpunkten `/*2*/` und `/*3*/`, d.h., am Ende der aufgerufenen Funktion und unmittelbar nach Ausführung der Funktion. Aus der Skizze sollte hervorgehen, warum der Aufruf von `testBumm` zu undefiniertem Verhalten führt. Kennzeichnen Sie freigegebenen Heap-Speicher.

c.) (4 P.) Die Klassenschnittstelle von `FileName` verfügt noch nicht über eine minimale Standard-schnittstelle. Erweitern Sie die Klasse, sodass die Deklaration der Klasse nun über eine minimale Standard-schnittstelle verfügt. Sie sollen hier *nur* die Deklaration der Klassenschnittstelle erweitern. Die Implementierung, d.h., die Definitionen der entsprechenden Methoden sollen in dieser Aufgabe **NICHT** erfolgen.

d.) (7 P.) Implementieren Sie nun den Kopier-Konstruktor der Klasse `FileName`.

e.) (7 P.)

Skizzieren Sie auf einem Extra-Blatt nach Implementierung des Kopier-Konstruktors nun nochmals die Speicherbelegung zum Zeitpunkt `/*2*/`, d.h., in der aufgerufenen Funktion. Aus der Skizze sollte hervorgehen, warum der Aufruf von `testBumm` dieses Mal zu einem **definierten** und erwarteten Verhalten führt.

f.) (2 P.) Zu welcher Ausgabe nach `datei` führt nach Ihrer erfolgreichen Verbesserung nun der Aufruf von `testBumm`?

g.) (4 P.) Implementieren Sie für die Header-Datei `FileName.h` einen Includewächter (nicht mit `pragma`). Kürzen Sie mit `/* ... */` ab; die Präprozessor-Anweisungen genügen also.

Aufgabe 2 : Objekterzeugung

ca. 15 Punkte

Implementieren Sie eine Überwachungsfunktionalität, d.h., die Klasse `Watchdog`, die beim Verlassen einer Funktion bzw. eines Blocks prüft, ob der Wert der überwachten Variablen gleich dem zweiten Wert der Konstruktor-Argumente ist. Der Aufruf der folgenden Funktion `anwWatchdog1`:

```
void anwWatchdog1() {
    FileName file("2021-01-18__11-32-14-00");
    int i = 15;
    // check whether at end year is equal to 2021
    Watchdog w1(file.GetYear(), 2021, "year");
    // check whether at end i is equal to 18
    Watchdog w(i, 18, "i");
    if (file.GetYear() > 10) {
        // check whether at end of block milli sec is 33
        Watchdog w2(file.GetMillisec(), 33, "msec");
        file.SplitFilenameIntoDate(
            "2021-12-18__12-32-14-33");
        return;
    }
    else {
        Watchdog w3(file.GetMillisec(), 11, "msec");
        file.SplitFilenameIntoDate(
            "2040-12-18__12-32-14-77");
    }
}
```

soll somit zur folgenden Bildschirmausgabe (oder Ausgabe in eine Datei) führen:

```
msec is 33 OK
i different from 18, now 15!!!
year is 2021 OK
```

Implementieren Sie nun die Klasse `Watchdog`, d.h., sowohl Header- als auch Quellcode-Datei. Sie dürfen natürlich auch alles `inline` im Header implementieren. Die Klasse `FunktionLog` aus der Vorlesung oder die STL-Klasse `unique_ptr` können als Idee für die Implementierung dienen.

Aufgabe 3 : STL Container und Lambda-Ausdrücke

ca. 21 Punkte

a.) (3 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `findIfVector` ?

```
void findIfVector () {
    vector<int> v = { 13, 1, 4 };
    auto iter = find_if (v.begin(), v.end(),
        [](int a) { return a > 2; });
    datei << (iter != v.end() ? *iter : 0);
}
```

b.) (4 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `findIfSet` ?

```
void findIfSet () {
    set<int, less<int>> s1 = { 13, 1, 4 };
    auto iter = find_if (s1.begin(), s1.end(),
        [](int a) { return a > 2; });
    datei << (iter != s1.end() ? *iter : 0) << " ";

    set<int, greater<int>> s2 = { 13, 1, 4 };
    auto iter2 = find_if (s2.begin(), s2.end(),
        [](int a) { return a > 2; });
    datei << (iter2 != s2.end() ? *iter2 : 0) << " ";
}
```

c.) (14 P.) Im folgenden Code sollen Sie drei Lambda-Ausdrücke zur Prüfung von Instanzen der Klasse `FileName` implementieren, sodass sich nach Aufruf von `findIfVectorFilename` die folgende Ausgabe nach `datei` ergibt:

```
Wrong: 18.13.2021
OK
Wrong: 32.1.2021
OK
Wrong: 31.4.2021
```

```
void findIfVectorFilename () {
    vector<FileName> v;
    v.push_back(FileName("2021-02-18__11-32-14-00"));
    v.push_back(FileName("2021-04-31__11-32-14-00"));
    v.push_back(FileName("2021-01-32__11-32-14-00"));
    v.push_back(FileName("2021-13-18__11-32-14-00"));
}
```

```
// checking of invalid month not in [1..12] with lambda
auto lambdaCheckMonth = []( /* Sie sind dran ... */ ;

// checking of invalid day not in [0..31] with lambda
auto lambdaCheckDaySimple = /* Sie sind dran ... */ ;

// checking of invalid day of a month with lambda
array<int, 12> dPM = { 31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31 };
// use dPM, April has only 30 days
auto lambdaCheckDayComplex = /* Sie sind dran ... */ ;
```

```
// checking of invalid month not in [1..12] with lambda
auto iter = find_if (
    v.begin(), v.end(), lambdaCheckMonth);
datei << (iter != v.end() ?
    "Wrong: " + iter->GetGermanDate() : "OK");
datei << "\n";
```

```
// checking of invalid month not in [1..12] with lambda
iter = find_if (
    v.begin(), ++v.begin(), lambdaCheckMonth);
datei << (iter != ++v.begin() ?
    "Wrong: " + iter->GetGermanDate() : "OK");
datei << "\n";
```

```
// checking of invalid day not in [0..31] with lambda
iter = find_if (
    v.begin(), v.end(), lambdaCheckDaySimple);
datei << (iter != v.end() ?
    "Wrong: " + iter->GetGermanDate() : "OK");
datei << "\n";
```

```
// checking of invalid day not in [1..31] with lambda
iter = find_if (
    v.begin(), ++v.begin(), lambdaCheckDaySimple);
datei << (iter != ++v.begin() ?
    "Wrong: " + iter->GetGermanDate() : "OK");
datei << "\n";
```

```
/* checking of invalid day of a month with lambda
April has only 30 days */
iter = find_if (
    v.begin(), v.end(), lambdaCheckDayComplex);
datei << (iter != v.end() ?
    "Wrong: " + iter->GetGermanDate() : "OK");
datei << "\n";
}
```

Ganz konkret: Definieren Sie die Variablen `lambdaCheckMonth`, `lambdaCheckDaySimple` und `lambdaCheckDayComplex`, d.h. weisen Sie die entsprechenden Lambda-Ausdrücke zu. Verwenden Sie Referenzen und `const`, wo möglich und sinnvoll, sodass **unnötiger Kopieraufwand** vermieden wird.