

3. Semester : 1. Prüfung V2

Name :	<input type="text"/>
--------	----------------------

- Die gesamte Prüfung bezieht sich auf die Programmierung in C++ !!
- Prüfungsdauer: 90 Minuten
- mit Kugelschreiber oder Tinte schreiben
- Lösungen können direkt auf die Aufgabenblätter geschrieben werden
- PCs sind nicht erlaubt
- Unterlagen und Bücher sind erlaubt
- Die Aufgaben sind in Fragen unterteilt, die mit Kleinbuchstaben gekennzeichnet sind
- Achte auch auf Details wie Punkte oder Kommas und Semikolons. !!!!!
- In den Beispielen fehlt meistens folgender Code, der als gegeben gilt:

```
#include <iostream>
using namespace std;
int main()
{
    ....
    return 0;
}
```

Aufgabe	Punkte	
Allgemeine Fragen	8	
Klassen	15	
Dynamischer Speicher	12	
Klassen und dyn. Speicher	15	
Klassen und Ableitungen	9	
TOTAL	59	

1. Verschiedene/allgemeine Fragen

- a) Wie heissen die beiden Hersteller der Entwicklungsumgebungen, die wir bis anhin kennegelernt haben? (2 Punkte)

Borland
Microsoft

- b) Nenne mindestens ein 32-Bit Betriebssystem. (1 Punkt)

Windows 2000

- c) Definiere eine Variable, die die aktuelle Jahreszahl enthält und zur Laufzeit nicht mehr geändert werden kann, also konstant bleibt. (2 Punkte)

```
const long Jahr = 2001;
```

- d) Wie viele Methoden hat folgende Klasse und wieviele Datenelemente ? (3 Punkte)
-

```
class Test
{
    public:
        void sayHello();
        int  sayGoodbye();
        int  m_n1;
    private:
        int sayNothing();
        int m_n2;
        char* m_pText;
};
```

3 Datenelemente, wovon eines public
3 Methoden

2. Klassen (Datenelemente/Methoden)

- a) Schreibe für die folgende Klasse "Daten" den korrekten und sauberen Konstruktor unter die Klasse. (2 Punkte)

```
class Daten
{
    public:
        Daten();
    private:
        double* m_pTest;
};
```

```
Daten::Daten
    :m_pTest(0)
{
}
```

- b) Schreibe für die folgende Klasse "Complex" den Vergleichsoperator `operator==()`. Dieser Operator muss `true` zurückgeben falls die Objekte gleichwertig sind und `false` im anderen Fall (5 Punkte).

```
class Complex
{
    public:
        Complex();
        bool operator==(const Complex& c);
    private:
        double m_x;
        double m_y;
};
```

```
bool Complex::operator==(const Complex& c)
{
    if(m_x == c.m_x && m_y == c.m_y)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

- c) Konstante Datenelemente einer Klasse müssen auf besondere Art initialisiert werden! Gegeben sei die folgende Klasse "Konto". Schreibe den Konstruktor für diese Klasse direkt unter die Klassendeklaration. Dabei soll die Konstante "TagImJahr" den Wert 360 haben und derZinsSatz 0.75 betragen. (4 Punkte)
-

```
class Konto
{
    public:
        Konto();
    private:
        const double ZinsSatz;
        const unsigned long TageImJahr;
};
```

```
Konto::Konto()
    :ZinsSatz(0.75),
    TageImJahr(360)
{
}
```

- d) Folgende zwei Klassen haben keine Defaultkonstruktoren. Um ein Objekt der Klasse "Song" zu erstellen muss man also den Songtitel und die Songlänge als Konstruktormparameter übergeben. Das gleiche gilt für die Klasse "Zeitdauer". Die Klasse Songtitel enthält ein Datenelement der Klasse "Zeitdauer" ! Schreibe den Konstruktor der Klasse Songtitel ! (4 Punkte)
-

```
class Zeitdauer
{
    public:
        // Kontruktor mit Parametern
        Zeit(double Sekunden);
    private:
        double m_Sekunden;
};

class Songtitel
{
    public:
        Songtitel(double Sekunden, string Titel);
    private:
        Zeitdauer    m_Zeitdauer;
        string      m_Titel;
};
```

```
Songtitel::Songtitel(double Sekunden, string Titel)
    :m_Zeitdauer(m_Zeitdauer), m_Titel(Titel)
{
}
```

3. dynamischer Speicher

a) Wieviel Bytes Speicher werden im folgenden Beispiel alloziert ? (1 Punkt)

```
double* pWerte = new double[33];
```

```
33 * 8 Bytes = 264;
```

b) Schreibe im folgenden Beispiel die Zeile Code auf, die es benötigt um den Speicher wieder freizugeben, der alloziert wird (2 Punkte).

```
char* pSz = new char[1];
```

```
delete [] pSz;
```

c) Schreibe wie im Beispiel b) den Code um den Speicher freizugeben, der alloziert wird (2 Punkte).

```
string* pString = new string;
```

```
delete pString;
```

- d) Schreibe den Code nieder, der **genau** das macht was verlangt wird !
- Erzeuge einen char*, der auf meinen Namen "Markus" zeigt (1 Punkt).
 - Alloziere auf dem Heap (nicht auf dem Stack, also dynamisch) Speicher um **alle** Zeichen, die im vorher erzeugten "Markus"-char-String vorhanden sind aufzunehmen (2 Punkte).
 - Kopiere nun alle Zeichen einzeln vom ersten char-String in den zweiten (3 Punkte).
 - Gib den dynamisch allozierten Speicher wieder frei (1 Punkt).
- Ich verwende die Bezeichnung char-String um diese vom der string Klasse zu unterscheiden.
(Total 7 Punkte)

```
char* pMarkus = "Markus";  
char* pDynMarkus = new char[7]; // 6 Zeichen plus abschliessende 0  
for(int i = 0; i < 7; ++i)  
{  
    pDynMarkus[i] = pMarkus[i];  
}  
delete [] pDynMarkus;
```

4. Klassen und dynamischer Speicher

- a) Betrachte folgende Klassendeklaration und -definition und die main-Funktion darunter. Wieviel Speicher wird am Ende nicht richtig freigegeben ? (3 Punkte)
-

```
class Data
{
    public:
        Data();
    private:
        char m_data[20];
        unsigned long m_used;
};

Data::Data()
{
}

int main()
{
    Data* p = new Data;

    return 0;
}
```

Ein ganzes Data-Objekt bestehend aus 20 chars (je 1 Byte) + 1 unsigned long (4 Byte)

24 Bytes

- b) Auch bei diesem nächsten Beispiel wird Speicher nicht freigegeben. Wieviel ? (2 Punkte)
-

```
class Fred
{
    public:
        Fred();
    private:
        double* m_p;
};
Fred::Fred()
{
    m_p = new double[10];
}

int main()
{
    Fred einFred;
    return 0;
}

10 * 8 Bytes = 80 Bytes
```

- c) Schreibe den korrekten Code der Funktion, die dieser Klasse fehlt. (2 Punkte)

```
Fred::~Fred()
{
    delete [] m_p;
}
```

- d) Hier ist noch ein main, das die Klasse Fred von der Aufgabe b) verwendet. Wieviel Speicher wird nicht freigegeben wenn die Klasse Fred unverändert aus b) übernommen wird ? (2 Punkte). Wieviel Speicher wird nicht freigegeben, wenn man die Klasse Fred mit der Korrektur aus c) verwenden würde ? (2 Punkte). (Total 4 Punkte)
-

```
int main()
{
    Fred einFred;
    Fred* nochEinFred = new Fred;
    return 0;
}
1. 160 Bytes
2. 80 Bytes
```

- e) Was gibt folgendes Programm aus ? Pass genau auf ! (Tipp : Wieviele WinkyDinky's werden erzeugt und wieviele entfernt ?) (4 Punkte)
-

```
class WinkyDinky
{
    public:
        WinkyDinky();
        ~WinkyDinky();

        void Winke();
};

WinkyDinky::WinkyDinky()
{
    cout << "Ich bin WinkyDinky" << endl;
}

WinkyDinky::~~WinkyDinky()
{
    cout << "Ich gehe jetzt" << endl;
}

void WinkyDinky::Winke()
{
    cout << "Ich winke" << endl;
}

int main()
{
    WinkyDinky winky;
    winky.Winke();
    WinkyDinky* nochEinWinky = new WinkyDinky;
    nochEinWinky->Winke();
    return 0;
}

Ich bin WinkyDinky
Ich winke
Ich bin WinkyDinky
Ich winke
Ich gehe jetzt
```

5. Klassen und Ableitungen

a) Gegeben sei die folgende Klasse "Daten" :

```
class Daten
{
    public:
        Daten();
        void SetzeDaten(double Wert);
    private:
        double m_Wert;
};
```

Schreibe den Code zur Funktion SetzeDaten, die zur Klasse "Daten" gehört, so dass nach dem Funktionsaufruf "m_Wert" den Wert des Parameters der Funktion hat (1 Punkt).

```
void Daten::SetzeDaten(double Wert)
{
    m_Wert = Wert;
}
```

Du willst für eine Messreihe diese Klasse erweitern. Eigentlich willst Du alles von der Klasse Daten verwenden, nur zusätzlich brauchst du einen Zeitpunkt, der als **double** die seit Messbeginn verstrichenen Sekunden enthält. Erweitere also die Klasse Daten indem Du von ihr ableitest und nenne diese Klasse "Messwert"! Schreibe nun die Klassendeklaration (also nur was im Header-File wäre, wie oben für die Klasse Daten) für diese Klasse Messwert (2 Punkte). Mit der Funktion "SetzeDatenUndZeit" gibt es noch einen Punkt (1 Punkt).

```
class Messwert : public Daten
{
    public:
        void SetzeDatenUndZeit(double Wert, double Zeit);
    private:
        double m_Zeit;
};

void Messwert::SetzeDatenUndZeit(double Wert, double Zeit)
{
    m_Wert = Wert;
    m_Zeit = Zeit;
}
```

- b) Im Diagramm unten haben wir die zwei Klassen "Uhr" und "Wecker". Die verwendete Notation zeigt an, dass die Klasse "Wecker" von der Klasse "Uhr" abgeleitet ist. Der Pfeil zeigt dabei von der abgeleiteten Klasse zur Elternklasse. Eine Klasse kann mehrere Elternklassen haben !

Ergänze nun das Diagramm mit folgenden Klassen und setze die Klassen in die richtige Beziehung zueinander, indem du herausfindest welche Klassen voneinander ableiten. Es kann sein, dass eine Klasse überhaupt nicht in Beziehung zu einer anderen Klasse steht. Zeichne diese einfach alleine auf. Klasse "Reisewecker", Klasse "Taschenuhr", Klasse "Armbanduhr", Klasse "Digitalarmbanduhr", Klasse "Lautsprecher", Klasse "Taschenwecker". (6 Punkte)

