3. Semester : 1. Prüfung

Name :	

- 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	6	
Klassen	14	
Dynamischer Speicher	11	
Klassen und dyn. Speicher	12	
Klassen und Ableitungen	8	
TOTAL	51	

1. Verschiedene/allgemeine Fragen

a) Wir kennen die Entwicklungsumgebung von Microsoft, nämlich das Visual Studio schon einige Zeit. Nenne einen weiteren Anbieter einer visuellen Entwicklungsumgebung. (1 Punkt)

Borland

b) Wie heisst das Betriebssystem, das wir meistens benützen. (Es genügt die grobe Bezeichnung, z.B. “Doors” für “Doors 3000”). (1 Punkt)

Windows

c) Definiere eine Variable, die meinen Namen “Markus” enthält und zur Laufzeit nicht mehr geändert werden kann, also konstant bleibt. (2 Punkte)

const char* markus = “Markus”;

d) Wie viele Methoden hat folgende Klasse und wieviele Datenelemente ? (2 Punkte)

```
class Test
{
    public:
        void funktion1();
        int  foo();
    private:
        int Bah();
        int m_blah;
        long m_Long;
};
```

3 Methoden, wovon ein privat ist (int Blah())
2 private Datenelemente

2. Klassen (Datenelemente/Methoden)

a) Schreibe für die folgende Klasse “Daten” den korrekten Konstruktor unter die Klasse. (2 Punkte)

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

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

b) Schreibe für die folgende Klasse “Complex” den operator- (minus). (4 Punkte)

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

```
void Complex::operator-(const complex& c)
{
  m_x = m_x - c.m_x;
  m_y = m_y - c.m_y;
}
```

c) Folgende zwei Klassen haben keine Defaultkonstruktoren. Um ein Objekt der Klasse “Messwert” zu erstellen muss man also die Zeit und den Wert als Konstruktorparameter übergeben. Das gleiche gilt für die Klasse “Zeit”. Die Klasse Messwert enthält ein Datenelement der Klasse Zeit ! Schreibe den Konstruktor der Klasse Messwert ! (4 Punkte)

```
class Zeit
{
    public:
        // Kontruktor mit Parametern
        Zeit(int Minuten);
    private:
        int m_Minuten;
};

class Messwert
{
    public:
        Messwert(int Minuten, double Wert);
    private:
        Zeit   m_Zeit;
        double m_Wert;
};
```

```
Messwert::Messwert(int Minuten, double Wert)
          :m_Zeit(Minuten),
           m_Wert(Wert)
{
}
```

d) Konstante Datenelemente einer Klasse müssen auf besondere Art initialisiert werden! Gegeben sei die folgende Klasse “Kreis”. Schreibe den Konstruktor für diese Klasse direkt unter die Klassendeklaration. Dabei soll die Konstante “dieEins” den Wert 1.0 haben. (4 Punkte)

```
class Kreis
{
    public:
        Kreis();
    private:
        const double pi;
        const double dieEins;
;
```

Kreis::Kreis()
pi(3.1415), dieEins(1.0)
{
}

3. dynamischer Speicher

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

```
char* text = new char[37];
```

37 Bytes

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

```
double* pDouble = new double;
```

```
delete pDouble;
```

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

```
char* pChar = new char[20];
```

```
delete []pChar;
```

d) Bei dieser Aufgabe musst Du ein wenig Code schreiben !
Alloziere auf dem Heap (nicht auf dem Stack, also dynamisch) Speicher für 33 double - Werte (2 Punkte). Setze danach diese 33 Werte alle auf 3.5 (3 Punkte). Gib danach den Speicher wieder frei (1 Punkt).
(Total 6 Punkte)

```
double* Werte = new double[33];
for(int i = 0; i < 33; ++i)
{
    Werte[i] = 3.5;
}
delete []Werte;
```

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 Complex
{
    public:
        Complex();
    private:
        double m_x;
        double m_y;
};

Complex::Complex()
{
}

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

    return 0;
}
```

Ein Complex Objekt, das zwei double-Werte à 8 Bytes enthält => 16 Bytes

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

```
class Fred
{
    public:
        Fred();
    private:
        char* m_p;
};
Fred::Fred()
{
    m_p = new char[50];
}

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

50 Bytes !

c) Was fehlt in der Klasse “Fred“ aus Beispiel b) damit sie den Speicher richtig aufräumen würde ? (1 Punkt) Der Code dafür ist erst in der Aufgabe d) gefragt !

Ein Destruktor

d) Schreibe den Code zu der Funktion, die im Beispiel b) noch fehlt. (2 Punkte)

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

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 = new WinkyDinky;
    winky->Winke();
    WinkyDinky nochEinWinky;
    return 0;
}
```

Ich bin WinkyDinky
Ich winke
Ich bin WinkyDinky
Ich gehe jetzt

5. Klassen und Ableitungen

a) Gegeben sei die folgende Klasse “Daten” und eine main - Funktion als Beispiel :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
class Daten
{
    public:
        void Einlesen(ifstream& in);
    private:
        string m_daten;
};

void Daten::Einlesen(ifstream& in)
{
    getline(in, m_daten, '\n');
}

int main()
{
    ifstream file("daten.txt");
    Daten dieDaten;
    dieDaten.Einlesen(file);
    return 0;
}
```

Was muss an der Klasse Daten geändert werden, damit sie mit der gleichen Funktion “Einlesen” sowohl aus einem File lesen als auch von der Tastatur einlesen kann. Du kannst entweder die Funktion “Einlesen” neu schreiben, oder aufschreiben was daran geändert werden muss. (2 Punkte)

Es genügt den Parameter “in” der Einlesen Funktion vom Datentyp ifstream auf den Datentyp istream zu ändern.

b) Im Diagramm unten haben wir die zwei Klassen “Säugetier” und “Tier”. Die verwendete Notation zeigt an, dass die Klasse “Säugetier” von der Klasse “Tier” abgeleitet ist. Der Pfeil zeigt dabei von der abgeleiteten Klasse zur Elternklasse.
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 “Insekt”, Klasse “Mensch”, Klasse “Hund”, Klasse “Sportwagen”, Klasse “Spinne”, Klasse “Mann”. (6 Punkte)