3. Semester : 2. Prüfung Fake

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**	
Grundlagen		
Objektorientierte Programmierung		
Dynamischer Speicher und Klassen		
Vererbung und Polymorphismus		
Weitere Fragen		
TOTAL		

1. Grundlagen

a) Finde die richtigen Variablendefinitionen, also solche die keine Compilerfehler erzeugen und bezeichne diese mit einem Haken. (3 Punkte)

Deklaration	**Richtig ?**
const char* t = “5”;	ok
int* test = 0;	ok
float note = “ “;	
char* = “200”;	
ok = true;	
long test = 300;	ok

b) Definiere einen Aufzählungstypen als enum mit dem Namen Werktage und den Elementen Montag, Dienstag, Mittwoch, Donnerstag und Freitag (5 Punkte)

```
enum Werktage
{
  Montag,
  Dienstag,
  Mittwoch,
  Donnerstag,
  Freitag
};
```

c) Definiere eine Funktion Swap, die zwei Variablen vom Datentypen **int** tauscht. (4 Punkte)

```
void Swap(int& a, int& b)
{
  int z = a;
  a = b;
  b = z;
}
```

2. Objektorientierte Programmierung

a) Hier eine Klasse MeineKlasse (seufz).

```
#include <string>
#include <iostream>

using namespace std;

class MeineKlasse
{
  public:
    MeineKlasse(int Repetitionen);
    ~MeineKlasse();
    void TueEtwas(const string& was);
  private:
    int m_Repetitionen;
};

MeineKlasse::MeineKlasse(int Repetitionen)
{
  cout << "Element erstellt" << endl;
  m_Repetitionen = Repetitionen;
}

MeineKlasse::~MeineKlasse()
{
  cout << "Element zerstoert" << endl;
}

void MeineKlasse::TueEtwas(const string& was)
{
  for(int i = 0; i < m_Repetitionen; ++i)
  {
    cout << was << endl;
  }
}
```

Schreibe eine vollständige **main**-Funktion, die durch **Verwendung der Klasse MeineKlasse** folgende Ausgabe erzeugt (Der Code sollte nicht mehr als 5~6 Zeilen lang sein) : (4 Punkte)t

```
Element erstellt
Arbeite
Arbeite
Arbeite
Element zerstoert
```

```
int main()
{
  string was(“Arbeite”);
  MeineKlasse ein(3);
  ein.TueEtwas(was);
  return 0;
}
```

b) Definiere eine Klasse Person.
Eine Person hat einen Namen, wohnt an einem Ort und hat ein Alter. Diese Klasse soll einen Konstruktor haben, dem als Parameter der Name übergeben wird. Zusätzlich soll die Klasse zwei Funktionen haben, mit denen jeweils der Ort (SetzeOrt) und das Alter gesetzt werden kann (SetzeAlter). Du kannst die Klassendefinition und die Implementation direkt untereinander schreiben (wie in Teilaufgabe “a)” die Klasse MeineKlasse. ca. 20 Zeilen) (8 Punkte).

```
class Person
{
public:
  Person(string name);
  ~Person();
  void SetzeOrt(const string& ort);
  void SetzeAlter(int alter);
private:
  string m_Ort;
  int m_Alter;
};

Person::Person(const string& ort)
{
  m_Ort = ort;
}

Person::~Person()
{
}

void Person::SetzeOrt(string ort)
{
  m_Ort = ort;
}

void Person::SetzeAlter(int alter)
{
  m_Alter = alter;
}
```

3. Dynamischer Speicher und Klassen

a) Ergänze in folgender Klassedeklaration und -definition den Destruktor (2 Punkte) und den Code im Konstruktor, der alle Elemente im Array auf 0 initialisiert (2 Punkte). (Total 4 Punkte)

```
class Leck
{
public:
  Leck();
  // HIER DESTRUKTOR DEKLARIEREN
  ~Leck();


private:
  long* m_pData;
};

Leck::Leck()
{
  m_pData = new long[100];
  // HIER DATEN IM ARRAY AUF 0 INITIALISIEREN
  for(int i = 0; i < 100; ++i)
  {
    m_pData[i] = 0;
  }

}

// HIER KORREKTE DESTRUKTORDEFINITION EINFÜGEN
Leck::~Leck()
{
  delete []m_pData;
}
```

b) Ergänze in der Klasse unten den Code für den Destruktor (2 Punkte) und den Kopierkonstruktor (4 Punkte). (Total 6 Punkte)

```
#include <string.h>
class Daten
{
public:
  Daten(int* daten, int anzahl);// Konstruktor
  Daten(const Daten& c);// Kopierkonst.
  ~Daten(); // Destruktor
private:
  int* m_pDaten;
  int m_Anzahl;
};

Daten::Daten(int* daten, int anzahl)
{
  m_Anzahl = anzahl;
  m_pDaten = new int[m_Anzahl];
  for(int i = 0; i < m_Anzahl; ++i)
  {
    m_pDaten[i] = daten[i];
  }
}

Daten::~Daten()
{
  delete [] m_pDaten;
}


Daten::Daten(const Daten& c)
{
  m_Anzahl = c.m_Anzahl;
  m_pDaten = new int[m_Anzahl];
  for(int i = 0; i < m_Anzahl; ++i)
  {
    m_pDaten[i] = c.m_pDaten[i];
  }
}
```

4. Vererbung und Polymorphismus

a) Was gibt folgendes Programm aus ? (5 Punkte)

```
class Mutter
{
public:
  Mutter();
  ~Mutter();
};

class Kind : public Mutter
{
public:
  Kind();
  ~Kind();
};

Mutter::Mutter()
{
  cout << "Mutter kommt" << endl;
}

Mutter::~Mutter()
{
  cout << "Mutter geht" << endl;
}

Kind::Kind()
{
  cout << "Kind kommt" << endl;
}

Kind::~Kind()
{
  cout << "Kind geht" << endl;
}

int main()
{
  Mutter* p = new Kind;
  delete p;
  return 0;
}
```

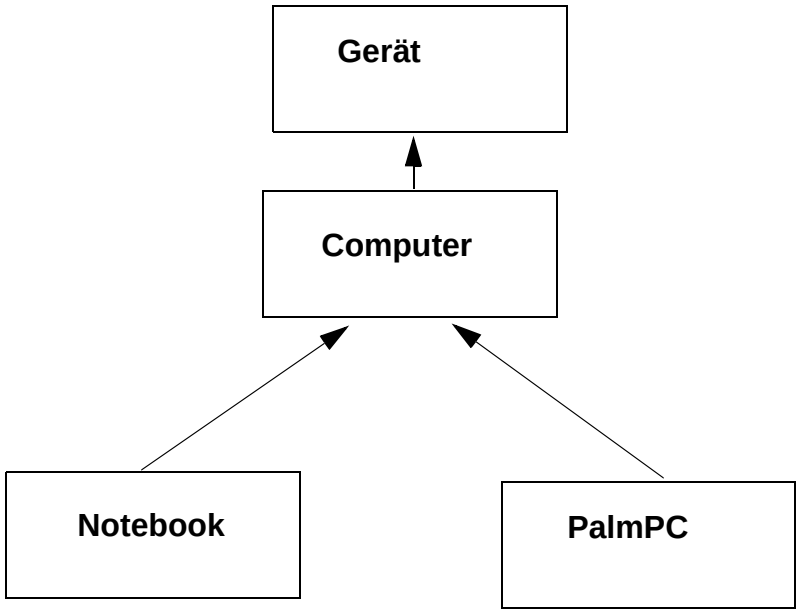
p ist ein Basisklassenzeiger ! Also Destruktor von Kind wird nicht aufgerufen, da der Destruktor nicht virtuell ist.

Ausgabe:

Mutter kommt
Kind kommt
Mutter geht

5. Weitere Fragen

a) Zeichne ein kleines Klassendiagramm mit den Beziehungen zwischen den Klassen, wie wir das bereits früher gemacht haben und zwar für folgende Klassen : Gerät, Computer, Notebook und PalmPC. (4 Punkte)



b) Willst Du ein guter C++ - Programmierer werden ? (1 Punkt)

JA