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
- PC's sind nicht erlaubt
- Unterlagen und Bücher sind erlaubt
- Achte auf Details wie Punkte, Kommas und Semikolons

Aufgabe	Punkte	
Dynamischer Speicher mit new, delete		
Zeiger und Vektoren		
Operatoren überladen		
Programmverständnis und Vererbung		
Verschiedene Fragen		
Total		

1. Dynamischer Speicher mit new und delete

a) Ergänze bei folgenden Beispielen den korrekten Aufruf von delete **falls dieser überhaupt nötig ist** (Vorsicht)! Bei den Beispielen wo eine delete unnötig ist, kannst du rechts daneben „NN“ schreiben. Ein Punkt pro Beispiel. (Total 5 Punkte)

```
int main()
{
    int* test = new int;
    *test = 23;

    delete test;

    return 0;
}
```

```
int main()
{
    double* werte = new double[20];

    delete [] werte;

    return 0;
}
```

```
int main()
{
    char a = ´a´;
    char* blah = &a;
```

NN

```
    return 0;
}
```

```
int main()
{
    int* pn = 0;
    pn = new int;

    delete pn;


    return 0;
}
```

```
int main()
{
    double d = 3.44;
    double* pd = &d;
    pd = new double;

    delete pd;

    return 0;
}
```

b) Erzeuge ein Array von 10 *double* mittels *new* und lösche dieses Array gleich wieder (2 Punkte)

```
double* p = new double[10];
delete [] p;
```

c) Erzeuge ein Array von 10 Zeigern auf *int*(int*). (1 Punkt)
Schreibe eine Schleife in der jeder Zeiger in diesem Array mit new initialisiert wird. (2 Punkte)
Schreibe eine zweite Schleife in der du den Speicher wieder freigibst. (2 Punkte) (Total 5 Punkte)

```
int* ZeigerArray[10];

for(int i = 0; i < 10; ++i)
{
    ZeigerArray[i] = new int;
}

for(int i = 0; i < 10; ++i)
{
    delete ZeigerArray[i];
}
```

2. Zeiger und Vektoren

a) Gegeben sei der C-string unten. Schreibe eine Schleife, die diesen **rückwärts** ausgibt, indem du die Zeichen einzeln aus dem Array holst und ausgibst! (4 Punkte)

```
#include <iostream>
using namespace std;

int main()
{
    char cstring[6] = „Hallo“;

    // Hier „Rückwärts“-Schleife einfügen



    for(int i = 4; i >= 0; --i) // index 5 enthält das
    {                           // abschliessende null
        cout << cstring[i];
    }



    return 0;
}
```

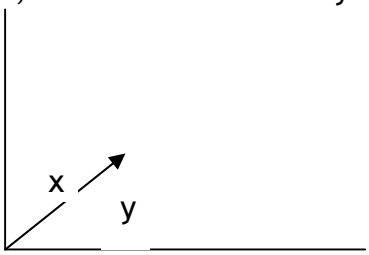
b) Was gibt folgendes Beispiel aus? (2 Punkte)

```
#include <iostream> // für cout
#include <string.h> // für strlen
using namespace std;

int main()
{
    char* text = „Er hat sich immer sehr angestrengt“;
    char* variante = „meist“;

    for(int i = 0; i < strlen(variante); ++i)
    {
        text[i+12] = variante[i];
    }

    cout << text;
    return 0;
}
```

```
Er hat sich meist sehr angestrengt
```

3. Operatoren überladen

a) Unten findest du eine Klasse *Vektor2D*. Ein solcher Vektor wird in der Geometrie und Mathematik, aber auch in der Elektronik etc. verwendet. Es handelt sich hierbei um eine Gerade, die eine Richtung und eine Länge hat, die mit einem x und y Wert definiert werden.



Man kann Vektoren addieren indem man jeweils die x-Werte und die y-Werte addiert. [2, 4] + [3, 4] = [5, 8].
Ergänze die Klasse *Vektor2D* mit einem +-operator (In der Klassendeklaration und in der Definition)! (4 Punkte)

```
class Vektor2D
{
    public:
        // Konstruktor
        Vektor2D(int x, int y);

        // Hier operator+ deklarieren!


        Vektor2D operator+(const Vektor2D& addend);


    private:
        int m_x;
        int m_y;
};

Vektor2D::Vektor2D(int x, int y)
         :m_x(x), m_y(y)
{
}

// Hier den operator-+ implementieren
Vektor2D Vektor2D::operator+(const Vektor2D& addend)
{
    // Wir geben als Ergebnis ein neues Objekt Vektor2D zurück
    Vektor2D neuesObjekt(0,0);

    neuesObjekt.m_x = m_x + addend.m_x;
    neuesObjekt.m_y = m_y + addend.m_y;

    return neuesObjekt;
}
```

b) Überlade in der folgenden Klasse *Person* den ==-operator (Vergleichsoperator) und ergänze einerseits die Klassendeklaration (Header-Datei) und die Definition (.cpp-Datei) (4 Punkte)

```
#include <string>
using std::string;

class Person
{
    public:
        Person(const string& vorname, const string& nachname);

        // Hier die Deklaration für den operator== einfügen


        bool operator==(const Person& anderePerson);



    private:
        string m_vorname;
        string m_nachname;
};

Person::Person(const string& vorname, const string& nachname)
       :m_vorname(vorname), m_nachname(nachname)
{
}

// Hier den Code für den operator== einfügen

bool Person::Person(const Person& anderePerson)
{
    bool sindGleich = false;
    if(m_vorname == anderePerson.m_vorname &&
       m_nachname == anderePerson.m_nachname)
    {
        sindGleich = true;
    }
}
```

4. Programmverständnis und Vererbung

a) Gegeben ist folgende Header-Datei zur Klasse *Person* (der Code zur Klasse Person, also die .cpp-Datei brauchst du nicht).
Schreibe eine kleine main-Funktion. Erzeuge ein Objekt dieser Klasse *Person*. Gib das Objekt dann mit der Funktion *Ausgabe* einerseits auf der Konsole, andererseits in eine Datei „Person.txt“ aus. Ergänze auch die nötigen #include und namespace-Angaben! (6 Punkte)

PERSON.H

```
#include <string>
#include <ostream>
using std::string;
using std::ostream;

class Person
{
    public:
        Person(const string& vorname, const string& nachname);
        void Ausgabe(ostream& outStream);
    private:
        string m_vorname;
        string m_nachname;
};
```

MAIN.CPP

```
// Hier #include-Anweisungen und namespace-Angaben einfügen

#include „Person.h“
#include <fstream>  // für ofstream
#include <iostream> // für cout

using namespace std;

int main()
{
    Person einePerson(„Fred“, „Feuerstein“);

    einePerson.Ausgabe(cout);
    ofstream datei(„Person.txt“);
    einePerson.Ausgabe(datei);

    return 0;
}
```

5. Verschiedene Fragen

a) Definiere eine Aufzählung (*enum*) Wochentage, die jedem Wochentag einen Wert zuweist, wobei der Montag den Wert 0 haben soll. (3 Punkte)

```
enum Wochentage
{
    Montag = 0,
    Dienstag,
    Mittwoch,
    Donnerstag,
    Freitag,
    Samstag,
    Sonntag
};
```

b) Erzeuge ein Array von 10 *int*-Werten (nicht dynamisch) und initialisiere es so, dass alle Elemente den Wert 0 haben auf **einer Zeile!** (2 Punkte)

```
int array[10] = { 0 };
```

c) Erzeuge mit *new* einen *double* Wert so, dass er den Wert 4.5 hat auf **einer Zeile!** (2 Punkte)

```
double* pd = new double(4.5);
```