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	14	
Zeiger und Vektoren	10 + 1	
Operatoren überladen	8	
Programmverständnis und Vererbung	10	
Verschiedene Fragen	8	
Total	50 + 1	

1. Dynamischer Speicher mit new und delete

a) Ergänze bei folgenden 5 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)

```
// Beispiel 1
int main()
{
    double fred = 5;
    double* pFred = &fred;

    kein new also auch kein delete nötig

    return 0;
}
// Beispiel 2
int main()
{
    char* s = new char[20];

    delete [] s;

    return 0;
}
// Beispiel 3
int main()
{
    char a = ´a´;
    char* blah = &a;
    blah = new char('b');

    delete blah;

    return 0;
}
// Beispiel 5
int main()
{
    int* test = new int[50];

    delete [] test;

    return 0;
}
// Beispiel 5
int main()
{
    float f = 1.5;
    float* pf = &f;

    kein new also auch kein delete nötig
    return 0;
}
```

b) Erzeuge ein Array von 5 Zeigern auf *char* (char*). (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)

```
char* array[5]; // Wir erzeugen also fünf Zeiger auch char

for(int i = 0; i < 5; ++i)
{
    array[i] = new char;
}
for(int j = 0; j < 5; ++j)
{
    char* zeiger = array[i]; // damit es klarer ist schreibe
    delete zeiger;           // ich den Code auf zwei Zeilen
}
```

c) Erzeuge ein Array von 20 doubles dynamisch. (1 Punkt)
Schreibe eine Schleife in der jedes Element den **Wert 3.5** erhält. (2 Punkte)
Lösche das Array wieder. (1 Punkt) (Total 4 Punkte)

```
double* array = new double[20]; // beachte den Unterschied zu
                                // oben !!
for(int i = 0; i < 20; ++i)
{
    array[i] = 3.5;
}

delete [] array;
```

2. Zeiger und Vektoren

a) Was gibt folgendes Programm aus? (2 Punkte)

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

const char* Worte[] =
{
    "Blah",
    "Der",
    "Die",
    "Das",
    "Emma",
    "Amma",
    "am",
    "Sepp",
    "Fred",
};

int indices[6] =
{
    3,
    2,
    0,
    4,
    5,
    2
};

int main()
{
    for(int i = 0; i < 5; ++i)
    {
        int index = indices[i];
        cout << Worte[index];
    }

    return 0;
}
```

```
DasDieBlahEmmaAmma
```

b) Schreibe eine Funktion *stringlen,* die als Rückgabewert die Anzahl Buchstaben in einem C-String berechnet, den man als Argument übergibt. Gegeben ist auch der Funktionsprototyp.
Tipp : Definiere eine Variable für die Länge, initialisiere diese auf 0 und mache eine Schleife, in der du diese erhöhst. In der Schleifenbedingung prüfst du ob der aktuelle Buchstabe (mit der aktuellen Länge als Index) ungleich 0 ist. Die Lösung ist etwa 6 Zeilen lang. (4 Punkte)

```
// Funktionsprototyp
int stringlen(const char* text);

// Hier bitte ganze Funktion hinschreiben (mit Funktionskopf
// wie beim Funktionsprototypen


int stringlen(const char* text)
{
    int laenge = 0;
    while(text[laenge] != 0)
    {
        ++laenge;
    }
    return laenge;
}
```

c) Schreibe eine Funktion *output*, die als Argument einen C-String hat und als zweites Argument einen *int*, der angibt wieviele einzelne Zeichen die Funktion ausgeben soll. Die Funktion hat keinen Rückgabewert (also *void*), als erstes Argument einen *const char** und als zweites Argument einen *int*.
Tipp : Schreibe eine Schleife, mit einer Zählvariable *i*, die als Index im C-String Argument verwendet wird.
Einen Zusatzpunkt kannst du die ergattern, wenn du prüfst, ob der character gleich 0 ist und die Schleife entweder mit *break* oder *return* verlässt. (4 Punkte + 1 Zusatzpunkt)

```
void output(const char* text, int laenge)
{
    for(int i = 0; i < laenge; ++i)
    {
        if(text[i] == 0) // abschliessende null ?
        {
            break;       // damit wird die umgebende for oder
        }                // while Schleife verlassen, aber auch
                         // ein switch Anweisung
        cout << text[i];
    }
}
```

3. Operatoren überladen

a) Du findest unten die Klasse *Note*, die als Datenelement einen *double* enthält. Überschreibe für diese Klasse den *operator+=*. Dieser Operator soll keinen Rückgabewert (also void) haben und hat als Argument einen konstante Referenz auf *Note*.
Ergänze zuerst die Klassendefinition mit der Methodendeklaration für diesen Operator (1 Punkt) und schreibe danach unter die Klasse die ganze Methode. (3 Punkte) (Total 4 Punkte)

```
class Note
{
public:
    // Konstruktor
    Note(double init);

    // Hier Methodendeklaration einfügen

    void operator+=(const Note& andere);



private:
    double m_wert;
};


// Konstruktorcode
Note::Note(double init)
     :m_wert(init)
{
}


// Hier Methodendefinition einfügen

void Note::operator+=(const Note& andere)
{
    m_wert += andere.m_wert;
}
```

b) Überlade in der folgenden Klasse *Punkt* den ==-operator. Beachte, dass diese Klasse zwei Datenelemente enthält, die gleich sein müssen, damit der Vergleichsoperator *true* zurückgibt. Ergänze auch hier die Klassendefinition (Header-Datei)(1 Punkt) und darunter den Code, der üblicherweise in der .cpp-Datei steht. (3 Punkte) Achte auf korrekte Parameterübergabe (Referenzen und const, etc.)
(4 Punkte)

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

    // Hier operator== deklarieren

    bool operator==(const Punkt& anderer);



private:
    int m_x;
    int m_y;
};


// Konstruktorcode
Punkt::Punkt(int x, int y)
      :m_x(x), m_y(y)
{
}



// Hier den Code für den operator== einfügen

bool Punkt::operator==(const Punkt& anderer)
{
    // es müssen beide Datenelemente übereinstimmen
    if(m_x == anderer.m_x && m_y == anderer.m_y)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

4. Programmverständnis und Vererbung

a) Gegeben ist folgende Header-Datei zur Klasse *Punkt*. Definiere eine Methode *Ausgabe*, die keinen Rückgabewert hat (void). Die Methode *Ausgabe* soll in ein beliebiges ostream-Objekt (als Argument) ausgeben können. Die Ausgabe soll etwa so aussehen:
x:45 y:56
(4 Punkte)

PUNKT.H

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

class Punkt
{
public:
    Punkt(int x, int y);

    // Hier Methode Ausgabe deklarieren

    void Ausgabe(ostream& out) const;



private:
    int m_x;
    int m_y;
};
```

PUNKT.CPP

```
Punkt::Punkt(int x, int y)
      :m_x(x), m_y(y)
{
}


// Hier den Code zur Methode Ausgabe ergänzen

void Punkt::Ausgabe(ostream& out)
{
    out << „x:“ << m_x;
    out << „y:“ << m_y;
    out << endl;
}
```

b) Gegeben ist folgende Klasse *Name*. **Leite** von dieser Klasse eine Klasse *Adresse* **ab**, die zusätzlich die Datenelemente Strasse und Ort enthält (wähle einen geeigneten Datentypen). Diese Klasse soll nur einen Konstruktor und die zwei zusätzlichen Datenelemente haben. Der Konstruktor der Klasse *Adresse* soll als Argumente den Vornamen, den Nachnamen, die Strasse und den Ort haben. Es genügt die korrekte Klassendeklaration. (4 Punkte)

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

class Name
{
public:
    // Konstruktor
    Name(const string& vorname, const string& nachname);

private:
    string m_vorname;
    string m_nachname;
};

// Hier die Klasse Adresse von Name abgeleitet definieren
// mit dem Konstruktor und zwei Datenelementen


class Adresse : public Name
{
    public:
        Adresse(const string& vorname, const string& nachname,
                const string& strasse, const string& ort);
    private:
        string m_strasse;
        string m_ort;
};
```

c) Welche Klasse ist im folgenden Beispiel die Basisklasse? (1 Punkt)

```
class A
{};
class B : public A
{};
```

A

d) Und welche Klasse erbt von welcher Klasse? A von B oder B von A (1 Punkt)

B von A

5. Verschiedene Fragen

a) Wandle folgende for-Schleife in eine while-Schleife um. (Stichworte: Initialisierung, Schleifenbedingung, Reinitialisierung) (3 Punkte)

```
for(int i = 0; i < 10; ++i)
{
    cout << i << endl;
}
```

```
int i = 0;
while(i < 10)
{
    cout << i << endl;
    ++i;
}
```

b) Definiere eine **Konstante** PI mit dem Wert 3.1415. Wähle den richtigen Datentypen. (2 Punkte)

```
const double PI = 3.1415;
```

c) Erzeuge mit *new* einen *char* Wert so, dass er den Wert ‚A' hat auf **einer Zeile!** Gib den Speicher auf einer zweiten Zeile wieder frei. (2 Punkte)

```
char* wert = new char(‚A');
delete wert;
```

d) Willst du fehlerfreie Programme schreiben? (1 Punkt)

```
Ja. Frag nicht immer so blöd
```