Gedächtnisprotokoll Grundlagen C++ (WiSe 23/24)

Aufgabe 1

Mehrere Klassen und außerhalb der Klassen zwei Initialisierungen der Klassen (Pointer und Referenz) gegeben

1. Zugriffe auf Klassenattribute (über Initialisierung 1)
2. Zugriffe auf Klassenattribute (über Initialisierung 2)
3. Angaben von Typen von Ausdrücken, teilweise mit Fehlern

Aufgabe 2

Vererbungen (ähnlich wie in Aufgabe 2 in der Altklausur von 2013) waren gegeben:

```
class Pokemon {
  public:
    ~Pokemon() { ... some print statement ... }
    virtual void attack() = 0;
}

class Pikachu : public Pokemon {
  public:
    Pikachu() { ... some print statement ... }
    void attack() { ... some print statement ... }
}

class Charmander : public Pokemon {
  public:
    ~Charmander() { ... some print statement ... }
    void attack() { ... some print statement ... }
}

class Eevee : public Pokemon {
  public:
    void attack() { ... some print statement ... }
}

class Vaporeon : public Eevee {
  public:
    ~Vaporeon() { ... some print statement ... }
    void attack() { ... some print statement ... }
}

int main() {
  Pokemon* vaporeon = new Vaporeon(); vaporeon->attack();
  Charmander* charmander = new Charmander(); charmander->attack();
  Eevee* eevee = new Eevee();

  delete vaporeon;
  delete charmander;
  delete eevee;
}
```

- a) Welche Klassen sind abstrakt? Wie wird das in C++ ausgedrückt?
- b) Bestimmen, welche Ausgabe in Konsole erfolgt (anhand davon welche Methoden oder Destruktoren überschrieben wurden)
- c) Ausgabe der main war unerwartet - Wieso? (`virtual` keyword in Basisklasse hat gefehlt)
- d) Welchen Typ hat das jeweils folgende Objekt:
  - ein `dynamic_cast<...>` innerhalb des gleichen Teilbaums der Vererbungshierarchie (also z.B. Vaporeon auf Pokemon)

    - ein `dynamic_cast<...>` über Grenzen des gleichen Teilbaums der Vererbungshierarchie hinweg (also z.B. Charmander auf Pikachu)
- e) `override` an allen möglichen, sinnvollen Stellen hinzufügen

Aufgabe 3

```
int foo(){
    return 42;
}
// TODO: Typedef für FUNC
// TODO: Typedef für ARRAY

void foo(){
    FUNC f = foo;
    auto b = f(); // TODO: welchen type hat auto

    ARRAY c = {1,2,3};

    int r = c[0];
    int& s = c[1];
    int* t = &c[2];

    r=s=*t = 0; //Wie sieht Array danach aus?

    // vergessen ...

    auto p = "hello world";
    p = p + 6;
    auto q = *p;  // TODO: Type und Wert von q;

    int* z = new int[4];
    // TODO free z;
}
```

Code gegeben,

- gibt die Typdeklaration für `FUNC` und `ARRAY` an
- Welchen Wert haben folgende Variablen: ... ?
- Wie kann man `z` löschen?

Aufgabe 4

Speicherblock und Code, der Pointer Arithmetik nutzt, waren gegeben

- Speicher am Ende des Programms (ähnlich wie in Aufgabe 4 der Altklausur von 2013) sollte ausgefüllt werden (diesmal sogar einfacher, ohne Überschreiben)

Aufgabe 5 (*const*)

C++ Code gegeben mit Lücken: `const` an allen Stellen ergänzen wo dies möglich ist (ähnlich wie Aufgabe 7 in der Altklausur von 2013).

Aufgabe 6

C++ Code mit Memory Leaks war gegeben.

- a) Memory Leaks durch das Ergänzen von `delete` oder `delete[]` an den richtigen Stellen lösen
- b) Memory Leaks durch das Verwenden von `unique_ptr<...>` beheben

Aufgabe 7 (*Funktoren*)

Gegeben war eine Klasse `Student` und ein Funktionspointer für `std::sort` um zu sortieren.

- Schreibe den Aufruf von `std::sort` mit einem Lambda.
- Schreibe einen Funktoren als Klasse und nutzen diesen für den Aufruf von `std::sort`.
- Das dritte Argument von `std::sort` ist optional und sortiert per Default mit `<`. Implementiere `operator<` für die Klasse.

In dem originalen Quellcode gab es zusätzlich eine Zählvariable `count`.

- Schreibe den Lambda-Aufruf, sodass die Vergleiche gezählt werden.
- Implementiere den Funktor, sodass die Vergleiche gezählt werden und nutze diesen für den Aufruf von `std::sort`.

Aufgabe 8 (*Templates*)

```
template<unsigned int N, typedef T>
class Vec{
    T elements[N];
public:
    Vec();
    ~Vec();

    Vec<N,T> operator+(const Vec<N,T>& rhs){
        Vec<N,T> out;
        for(unsigned int i = 0; i < N; i++){
            out[i] = elements[i] + rhs[i];
        }
        return out;
    }

    Vec<N,T> operator-(const Vec<N,T>& rhs){
        Vec<N,T> out;
        for(unsigned int i = 0; i < N; i++){
            out[i] = elements[i] - rhs[i];
        }
        return out;
    }

    T& operator[](size_t i){
        return elements[i];
    }
}
```

Obiger C++ Code, der Templates verwendet war gegeben.

- Nutze `typedef` um den Typen `Vec3f` zu definieren, der Vektor für $\mathbb{R}^3$ beschreibt.
- Erkläre, wieso die folgenden Typen Probleme bereiten:
  - `Vec<5, std::string>`
  - `Vec<2, int*>`
- Folgender Code beschreibt das Skalarprodukt zweier Vektoren. Warum geht das so nicht? (Was fehlt in der Klasse, denn `dot` ist korrekt?)

```
template<unsigned int N, typename T>
T dot(const Vec<N, T>& a, const Vec<N, T>& b){
    T sum(0);
    for(unsigned int i = 0; i < N; i++){
        sum += a[i] * b[i];
    }
    return sum;
}
```

Aufgabe 9

```
void inverse(const DynFloatArr& a, const DynFLoatArr& b) {
    for (std::size ) {
        // ????
    }
}
int main(){
    DynFloatArr a;
    DynFloatArr b;
    a.push_back(1.0);
    a.push_back(1.1);
    b.push_back(1.2);
    DynFloatArray c = a+b;
    b = a;

    inverse(a, b);

    for (unsigned int i = 1; i < a.size(); i++) {
        a[i] = a[i-1];
    }
    assert(a >= 0.1f && "Message");
}

class DynFloatArray{
    // some Membervariablen
public:
    void push_back(float val);
    size_t size() const;
    // TODO: ...
}
```

(Code sinngemäß)

Gib die Signaturen (Konstruktor, Destruktor, Methoden) an, um den Quellcode funktionsfähig zu machen (ohne Definition). Achte auf `const` -Correctness. Insgesamt fehlen 8 Signaturen.

Aufgabe 10

Wo tritt der Fehler am ehesten auf (Preprozessor, Compiler, Runtimefehler oder Undefiniedbehavior, Linkerfehler)

- Include-Guards/ `#pragma once` vergessen
- Verschiedene Versionen einer Bibliothek eingebunden
- uninitialisierte static Membervariable eine Klasse
- überladene Methoden die sich nur in Return-type unterscheiden

- Zuweisung einer const variable
- Zugriff auf negativen Index eines Arrays
- uninitialisierter Pointer
- ...