UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

9. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Aufgabe 1: Funktionals (30 Punkte)

Betrachten Sie folgendes Code-Fragment, welches Sie in StudIP finden:

```
vector<int> v = {1, 4, 2, 8, 5, 7};

copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
cout << endl;

auto it = remove_if(v.begin(), v.end(),
                    bind(bind(equal_to<int>(),_1, 0),
                              bind(modulus<int>(), _1, 2)));

copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
cout << endl;

return 0;
```

Beantworten Sie folgende Fragen schriftlich:

1. Welches Problem soll mit dem vorgelegten Code-Fragment offensichtlich gelöst werden?
2. Erklären Sie die Signatur und Nutzung von `std::bind` im vorliegenden Beispiel. Wie werden die benutzten Funktionen und Argumente verknüpft?
3. Wenn Sie den Code ausführen, werden Sie sehen, dass die Ausgabe nicht den Erwartungen entspricht. Was muss geändert werden, damit die Ausgabe den Erwartungen entspricht.

Aufgabe 2: Integration von Smartpointern (70 Punkte)

Betrachten Sie die in Ihrem vorgegebene Musterlösung des letzten Aufgabenblatts. Wenn Sie sich die Allokation und Freigabe von dynamischem Speicher ansehen, werden sie bemerken, dass durch die Weitergabe von diversen Pointern die Speicherverwaltung im besten Fall als undurchsichtig zu bezeichnen ist (auch wenn lt. valgrind die von unserem Code benötigten Ressourcen wieder freigegeben werden). In dieser Aufgabe ersetzen wir alle Raw-Pointer durch Smartpointer.

Implementierung eines Smartpointers für C++-Arrays

Leider unterstützt C++ erst ab C++17 die Verwendung von Arrays in verwalteten Pointern. Um den Code mit C++11 kompatibel zu halten, implementieren Sie zunächst eine Klasse `shared_array` die geeignet von `std::shared_ptr` erbt. Der wesentliche Unterschied zwischen Pointern und Arrays in diesem Kontext ist, dass Arrays mittels `delete[]` freigegeben werden müssen, während einfache Zeiger mit dem Operator `delete` gelöscht werden. Implementieren Sie die Freigabe von Arrays, indem Sie in `shared_array` den Konstruktor der Oberklasse `std::shared_ptr` mit einem geeigneten Deleter aufrufen. Benutzen Sie dazu eine Lambda-Funktion. Damit – analog zu `std::shared_ptr` – ihre `shared_array`-Implementierung auch Zugriff auf die Array-Elemente ermöglicht, müssen Sie zusätzlich noch zwei Index-Operatoren für lesenden und schreibenden Zugriff hinzufügen.

Integration von Smartpointern

Nachdem Sie die Klasse `shared_array` implementiert haben, ersetzen Sie alle Raw-Pointer im vorgebenen Code (nicht in der 3DS-Bibliothek in `/ext`) durch Smartpointer und Smartarrays. Dazu einige Hinweise:

- Um sich die Deklaration der Templates zu vereinfachen, fügen Sie mittels einer geeigneten using-Deklaration in die relevanten Klassen einen inneren Typ `Ptr` hinzu, der einen Smartpointer auf die entsprechende Klasse repräsentiert.

Beispiel für die Klasse Texture:

```
class Texture
{
public:

    /// Type alias for managed pointers to Texture instanzes.
    using Ptr = std::shared_ptr<Texture>;

…

};
```

- Passen Sie die Signaturen der Material-Container in `Materials.hpp` geeignet an.
- Ersetzen Sie alle Index-, Vertex-, Normal- und sonstigen Buffer sowie die Pixelarrays durch geeignete `shared_arrays`. Definieren Sie dazu analog zum Texture-Beispiel passende Aliase.
- Eventuell müssen Sie an einigen Stellen zwischen Smartpointer-Repäsentationen downcasten. Dazu gibt es analog zum `static_cast`-Operator `std::static_pointer_cast`.

Abgabe:

Checken Sie Ihre Lösung des Aufgabenblattes bis Montag den 07.01.2019, 8:00 Uhr in den Master-Branch des git-Repositorys Ihrer Gruppe ein. Bringen Sie pro Gruppe einen Ausdruck des von Ihnen erstellten Codes mit.