Übung zur Vorlesung Programmieren 2
SoSe 2024

Blatt 6

Range-based for-loop, Lambdas, std::pair

Erfordert die VL bis Kapitel	Sequentielle Container
Abgabe bis	11. Juni 2024, 16:00 Uhr

Richtlinien für Abgaben:

Lösungen, die diese Richtlinien nicht erfüllen, werden mit *0 Punkten* gewertet!
Abgegeben werden alle vorgegebenen, oder für die Aufgaben neu erstellten, Code/Text-Dateien.
Abgaben erfolgen über den Git-Server (↗) des Instituts.
Jeder in der Gruppe ist für die (pünktliche) Abgabe mitverantwortlich. Dafür gilt nicht die Commit-Zeit, sondern der Push-Zeitpunkt auf dem Server.

In Git ...
- wird für jedes Übungsblatt ein neuer Ordner angelegt: *Gruppe_XxYZ/blatt1*, */blatt2*, */blatt3*, ...
- dürfen keine temporären/automatisch generierten Dateien hinzugefügt werden (→ *.gitignore*).
- wird nur auf den Server gepusht, wenn der Code im aktuellen Commit lauffähig ist.
  Nicht lauffähige lokale Commits (für Zwischenstände) sind OK und sogar empfohlen. Diese sollten allerdings mit `"WIP:"` (work in progress) am Anfang der Commit-Nachricht gekennzeichnet werden.

Es wird kein vollständiger Code-Style vorgegeben, aber ...
- innerhalb jeder Abgabe muss die Code-Formatierung konsistent sein.
- es wird sinnvoll und konsistent eingerückt.
- Variablen haben aussagekräftige Namen.
- Compiler-Warnungen im *eigenen* Code, die in der Konsole ausgegeben werden, gelten als Fehler.

Allgemeine Infos:

Die Aufgaben und Erklärungen orientieren sich an (C++17) CMake-Projekten in der CLion IDE (↗), für Studenten kostenlos verfügbar für Windows, macOS und Linux.

Richtlinien & Infos werden bei Bedarf angepasst/erweitert. Sie gelten entsprechend je Übungsblatt.

Hinweise zu Blatt 6:

Tests: Für einige Aufgaben gibt es wieder Tests, sodass geprüft werden kann, ob die entsprechenden Lösungen grundlegend korrekt sind. (Die Tests kompilieren erst, wenn die Lösungen implementiert sind.)

Aufgabe 1

10 Punkte

Range-based for-loops
"Normale" for-loops sind ein nützliches Konstrukt, um Operationen wiederholt auszuführen — z.B. um auf alle Elemente eines Vektors zuzugreifen.
Speziell für den Zugriff auf alle Elemente von Containern bietet C++ die sogenannten range-based-for-loops. Neben der Vermeidung von Tippfehlern führen range-based-for-loops auch zu übersichtlicherem Code, da zunächst ihre Syntax deutlich kürzer ist. Darüber hinaus liefern sie aber auch direkt (Referenzen auf) die Elemente in den Containern, statt nur den laufenden Index. Es entfällt also auch der Aufruf, um mit dem Index auf das entsprechende Element im Container zuzugreifen:

```
for (auto & elem : container) {
  elem.doSomething();
}
```

statt wie bisher:

```
for (unsigned int index = 0; index < container.size(); ++index) {
  auto & elem = container[index];
  elem.doSomething();
}
```

Ersetzt an allen geeigneten Stellen im Code die "normalen" for-loops durch range-based-for-loops. Achtet dabei für die Element-Variable auf die selben Richtlinien zur Nutzung von Kopien, Referenzen (`&`) und konstanten Referenzen (`const &`) wie bei Funktionsparametern.

Abgabe/Präsentation:

1. Alle zutreffenden (5) Stellen verwenden range-based-for-loops
2. Es wird jeweils die passende Deklaration der Schleifenvariablen (`const`, `&`) verwendet.

Aufgabe 2

9 Punkte

> *Lambdas als Funktionsparameter*
> Mithilfe von Lambdas können Funktionen wie einfache Variablen behandelt, und so z.B. auch als Funktionsparameter übergeben werden.

Implementiert die Funktionen `walk(..)` und `filter(..)` in der Klasse `Grid2D`. Beide Funktionen sollen jeweils eine Lambda-Funktion als Parameter annehmen.

In `walk(..)` sollen dieser Lambda-Funktion dann einfach nacheinander alle Elemente des Grids als Parameter übergeben werden, z.B. zur Ausgabe im Terminal:

```
1 Grid2D<int> grid(...
2 auto lambda1 = [](int const & gridElement) { cout << gridElement << " "; };
3 grid.walk(lambda1);
```

Die Funktion `filter(..)` funktioniert recht ähnlich. Sie soll die übergebene Lambda-Funktion ebenfalls auf alle Elemente anwenden. Allerdings soll die hier übergebene Lambda-Funktion einen `bool`-Wert zurückgeben. In `filter(..)` sollen dann alle Werte aus dem Grid in einem `vector` zurückgegeben werden, für die die übergebene Lamda-Funktion `true` ergibt, z.B. zum Filtern aller positiven Zahlen:

```
1 Grid2D<int> grid(...
2 auto lambda2 = [](int const & gridElement) { return gridElement > 0; };
3 auto positiveGridElements = grid.filter(lambda2);
```

Abgabe/Präsentation:

Die Template-Funktionen ...

1. sind korrekt deklariert und definiert.
2. wenden das übergebene Lambda auf alle Elemente im Grid an.
3. funktionieren korrekt.

Aufgabe 3

9 Punkte

Berechnet mithilfe von Lambda-Funktionen in *PlayerSea_stats.cpp* die verschiedenen Werte der Spielstatistik. Nutzt dazu `grid.filter(..)` in den beiden Funktionen `numShipPositionsUnhit(..)` und `numWaterPositionsHit(..)`. Nutzt dafür jeweils nur einen Aufruf von entweder `walk(..)` oder `filter(..)`.

Abgabe/Präsentation:

Vervollständigt die Funktionen ...

1. `numShipPositionsUnhit(..)`
2. `numWaterPositionsHit(..)`
3. `numMaxMissilesPerPosition(..)`

Hinweis: Die Warnung "grid: Unreferenzierter formaler Parameter" *in den 3 Funktionen entsteht durch fehlende Teile im vorgegebenen Code und sollte durch eure Implementierung verschwinden.*

Aufgabe 4

5 Punkte

> *std::pair*
> Die Klasse `std::pair` bietet ein sehr einfaches und intuitives Interface, um unkompliziert zwei Objekte zu kombinieren, z.B. als Rückgabewert von Funktionen.

Implementiert in *PlayerSea_stats.cpp* eine Funktion `numMultipleHits()`, die ermittelt wie viele Spielfeldpositionen mit einem Schiff von mehr als einer Rakete getroffen wurden, und wie viele Raketen es an diesen Stellen insgesamt sind. Nutzt dafür wieder nur einen Aufruf von `walk(..)` oder `filter(..)`. Sie soll, wie die vorherigen Funktionen, ein `OutputGrid` als Parameter erhalten, als Rückgabewert aber beide Werte liefern.

Abgabe/Präsentation:

1. Implementiert die Funktionen `numMultipleHits(..)`
2. Nutzt sie in `printStats()`

Aufgabe 5 (Bonus)

(3 Punkte)

Die Zähl-Funktionen in Aufgabe 3, die `grid.filter(..)` nutzen, brauchen eigentlich nur die Anzahl der gefilterten Elemente, und keinen `vector` mit den Elementen selbst.

Ergänzt `Grid2D` um eine Funktion `count(..)`, die analog zu `filter(..)` die Elemente mit einer übergebenen Lambda-Funktion prüft, passende Elemente aber nur zählt, ohne sie zu kopieren.

Ermittelt mithilfe dieser Funktion einen beliebigen weiteren (noch nicht vorhandenen) Wert für die Statistik und gebt diesen in `printStats()` aus.

Gesamt: 33 Punkte