Programmieren in C++

SS 2018

Vorlesung 11, Dienstag 10. Juli 2018

(Vererbung II und Projekt)

Axel Lehmann
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Ü10 — Vererbung
  - Evaluationsergebnisse — Nächste Woche
  - 橋をかけろ — Hashiwokakero
- Inhalt
  - Vererbung — override/final
  - Projekt — Problemstellung, Hintergründe
  - **Übungsblatt 11:** Grundgerüst des Projektes

Erfahrungen mit dem Ü10 1/2

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Nur Zeit gehabt für Aufgabe 0.
  - Ich habe die Evaluation gemacht.
    Bei einigen hat das mit der E-Mail nicht geklappt, hier schauen wir noch, was getan werden kann.
  - Der Zettel sah auf den ersten Blick ziemlich kompliziert aus, aber dank der Tests findet man sich doch ganz gut zurecht.
  - Einige hatten Probleme readFile/das Blatt allgemein zu verstehen
    Wieso wurde nicht (mehr) auf dem Forum gefragt?
  - explicit u.a.: V5 https://youtu.be/NfFSxZiOP1c?t=5305
    Ohne explizit kann der Compiler den Konstruktor benutzen ohne dass er explizit aufgerufen wird

Erfahrungen mit dem Ü10 2/2

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Im Japanischen haben sie für jedes Wort ein bestimmtes Zeichen
    Nicht ganz, aber es gibt 4 Schriften - siehe nächste Folie
  - Zur Frage: Hashiwokakero hat jetzt meine Sudoku-Sucht abgelöst :)
  - Hier ein Link zum Spiel:
    https://www.puzzle-bridges.com/
    Allerdings ist es nicht sehr gut implementiert, ich denke dass ich das in näherer Zukunft besser programmieren werde.

Exkurs - Japanische Schriften

- Hiragana
  - Silbenschrift für inländische Begriffe (gemischt mit Kanji verwendet), Partikel und Endungen, etc.
- Katakana
  - Wie Hiragana, aber für fremdländische Begriffe wie Namen (z.B. アクセル)
- Kanji
  - Etliche 1000 Zeichen, verschiedene Bedeutungen je Zeichen aber auch verschiedene Zeichen für „gleiche" Laute (verschiedene Betonungen)
    Z.B. hashi: 橋 (Brücke) und 箸 (Essstäbchen)
- Romaji (lateinische Schrift)

橋をかけろ (Hashiwokakero)

UNI FREIBURG

- Inselgruppe unverbundener Inseln
- Inseln befinden sich auf einem Gitter

- Brücken müssen gebaut werden
  - Dürfen nur auf Gitterlinien verlaufen
  - Nur gerade Brücken
  - Dürfen sich nicht kreuzen
  - Maximale Anzahl Brückenköpfe je Insel

- Spielbar z.B: https://de.puzzle-bridges.com/

Vererbung 1/4

UNI FREIBURG

- Problem
  - Kindklasse soll eine Funktion überschreiben

```
class Thing {
 public: virtual std::string toString() const;
};
```

  - Funktion in der Kindklasse hat jedoch eine andere Signatur

```
class StringThing : Thing {
 public: std::string toString(); // Compiles, but never called
}
```

Vererbung 2/4

UNI FREIBURG

- Lösung: override (seit C++11)
  - Kindklasse soll eine Funktion überschreiben

```
class Thing {
 public: virtual std::string toString() const;
};
```

  - In der Kindklasse markiert, dass eine Funktion überschrieben werden soll

```
class StringThing : Thing {
 public: std::string toString() override; // Compilererror
}
```

  - error: 'std::__cxx11::string StringThing::toString()' marked 'override', but does not override

Vererbung 3/4

UNI FREIBURG

- final (Klassen)
  - Es soll keine Kindklassen geben

```
class FinalThing final : Thing {
 public:
  std::string toString() const;
};
class OtherThing : FinalThing { … }; // Will not compile
```

  - error: cannot derive from 'final' base ‘FinalThing' in derivedtype ‘OtherThing'

Vererbung 4/4

UNI FREIBURG

- final (Funktionen)
  - Kindklassen sollen eine Funktion nicht überschreiben

```
class SemiFinalThing: Thing {
 public:
  std::string toString() const final;
  double multiply(double x, double y) const;
};
class AnotherThing : SemiFinalThing {
 public:
  std::string toString() const; // Will not compile
  double multiply(double x, double y) const override;
};
```

Overloading

UNI FREIBURG

- Gleicher Funktionsname, verschiedene Parameter
  - Verwand mit Templates
  - double multiply(double x, double y) {
    return x * y;
    }
  - double multiply(double x) {
    return x * x;
    }
  - std::string multiply(std::string& x, size_t y) {
    std::string r = x; for (size_t i = 1; i < y; ++i) { r += x;}
    return r;
    }
  - Man kann auch Operatoren wie =, <, >, [] überladen

Projekt 1/5

UNI FREIBURG

- Drei Projekte zur Auswahl
  - Projekt 1: 橋をかけろ (das Spiel)

    Puzzle einlesen und spielbar machen
  - Projekt 2: 橋をかけろ (automatischer Löser)

    Lösungsstrategien ausdenken und implementieren
  - Projekt 3: Thema eigener Wahl

    Von Umfang und Komplexität ähnlich zu Projekt 1 oder 2

    Projekt 3 nur für die, die fast alle Punkte aus Ü1 – Ü10 haben, und denen bisher alles sehr leicht fiel (nicht viele)

    Aber auch die können natürlich Projekt 1 oder 2 machen

Projekt 2/5

UNI FREIBURG

- Genaue Projektbeschreibungen auf dem Wiki
  - Insbesondere für jedes Projekt:

    Kurzbeschreibung

    Hintergrund

    Anforderungen (Minimum)

    Anforderungen (optional)

    Auf den folgendes Folien auch noch mal eine kurze

    Vorstellung + etwas Hilfestellung

Projekt 3/5

UNI FREIBURG

- Projekt 1: geforderte Funktionalität
  - Einlesen einer Instanz aus einer Datei (Formate siehe Wiki)
  - Konsolengrafik (in Farbe, Inseln mindestens 3x3 Zeichen)
  - Bedienung über Maus
  - Undo Funktion mit gegebener Obergrenze
  - Code nach den bisherigen Regeln
  - Weitere Details, siehe Beschreibung auf dem Wiki ... Im Zweifelsfall im Unterforum "Projekt" nachfragen
- Optionales:
  - Prüfen ob weniger Klicks möglich wären, Benutzerdefinierte Farben, Animiertes Brückenbauen, ...

Projekt 4/5

UNI FREIBURG

- Projekt 2: geforderte Funktionalität
  - Einlesen einer Instanz aus einer Datei (Formate siehe Wiki)
  - Implementieren von Grundlegenden vereinfachungen
  - Implementierung eines Lösers
  - Muss eine Mindestzahl an Instanzen lösen, oder erkennen, dass nicht lösbar.
  - Ausgabe der Lösung in zwei Formaten
  - Auch hier: Details auf dem Wiki + Fragen auf dem Forum

Projekt 5/5

UNI FREIBURG

- Design der .h Dateien für das Ü11
  - Keine Vorgaben/Vorlagen
  - Eventuell eine größere Herausforderung als bisher
  - Hinweise zur Umsetzung:

    Schreiben Sie zu jeder Funktion erst die **Dokumentation**

    Eine gute Funktion macht etwas **Intuitives**

    Eine gute Funktion lässt sich gut **testen**

    Eine gute Funktion wirft **nicht verschiedene Sachen** in einen Topf (zum Beispiel Spiellogik und Zeichnen)

    Probieren Sie verschiedene Designs aus

Ncurses - Farben

UNI FREIBURG

- Für Projekt 1:
  - // Initialize ncurses
    initscr();
    …
    start_color();
    init_pair(1, COLOR_BLACK, COLOR_RED);
    init_pair(2, COLOR_RED, COLOR_YELLOW);
  - // Usage of predefined colors
    attron(COLOR_PAIR(1));
    printw("This is black with a red background!\n");
    attron(COLOR_PAIR(2));
    printw(" This is red with a yellow background!\n");
    attroff(COLOR_PAIR(2));

Literatur / Links

- Referenzen
  - https://en.cppreference.com/w/cpp/language/override
  - https://en.cppreference.com/w/cpp/language/final
  - https://linux.die.net/man/3/color_pair