Übungsstunde Woche 10

Adel Gavranović adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Intro •00000

Self-Assessment III

Objekte

Overloading

Outro

Links

▶ polybox zum Material für die Übungsstunden

Follow-up aus vorheriger Übungsstunde

Follow-up aus vorheriger Übungsstunde

■ Vielen dank für das Feedback!



Allgemein

Intro

Allgemein

Intro

Anzahl Abgaben sind runter. Bitte versucht dranzubleiben! Die nächsten Themen sind unabhängig von den jetzigen, also Einsteigen ist recht einfach...

Allgemein

Intro

- Anzahl Abgaben sind runter. Bitte versucht dranzubleiben! Die nächsten Themen sind unabhängig von den jetzigen, also Einsteigen ist recht einfach...
- Empfehlung: auch unvollendete Aufgaben können abgegeben werden (hilft euch, da ich die Session dann in die Richtung lenken kann und auf Missverständnisse eingehen kann). Dort ist aber wichtig, dass ihr mir // als Kommentar mitteilt was genau nicht ging

Allgemein

Intro

- Anzahl Abgaben sind runter. Bitte versucht dranzubleiben! Die nächsten Themen sind unabhängig von den jetzigen, also Einsteigen ist recht einfach...
- Empfehlung: auch unvollendete Aufgaben können abgegeben werden (hilft euch, da ich die Session dann in die Richtung lenken kann und auf Missverständnisse eingehen kann). Dort ist aber wichtig, dass ihr mir // als Kommentar mitteilt was genau nicht ging

E8:T1 "Vector and Matrix Operations"

■ Idealerweise ist eure main() einfach nur noch ein Funktionsaufruf nach dem anderen

Allgemein

Intro

- Anzahl Abgaben sind runter. Bitte versucht dranzubleiben! Die nächsten Themen sind unabhängig von den jetzigen, also Einsteigen ist recht einfach...
- Empfehlung: auch unvollendete Aufgaben können abgegeben werden (hilft euch, da ich die Session dann in die Richtung lenken kann und auf Missverständnisse eingehen kann). Dort ist aber wichtig, dass ihr mir // als Kommentar mitteilt was genau nicht ging

E8:T1 "Vector and Matrix Operations"

- Idealerweise ist eure main() einfach nur noch ein Funktionsaufruf nach dem anderen
- Versucht Initialization erst im allerletzten Moment zu machen. Gibt es einen Grund, wieso das die meisten nicht gemacht haben?

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

■ const std::string&

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

- const std::string&
- führt keine Variablen ein, wenn es nicht nötig ist (z.B. wenn man eine Zahl nur einmal verwendet, braucht man dafür wirklich keinen Namen)

E8:T2 "Decode Binary NZZ Front Page"

■ const std::string&

Intro

führt keine Variablen ein, wenn es nicht nötig ist (z.B. wenn man eine Zahl nur einmal verwendet, braucht man dafür wirklich keinen Namen)

E8:T3 "Recursive Function Analysis"

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

- const std::string&
- führt keine Variablen ein, wenn es nicht nötig ist (z.B. wenn man eine Zahl nur einmal verwendet, braucht man dafür wirklich keinen Namen)

E8:T3 "Recursive Function Analysis"

■ seid spezifischer. nicht "es" sondern n ist streng monoton fallend, weil Es wäre doof, deswegen Punkte liegen zu lassen

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

- const std::string&
- führt keine Variablen ein, wenn es nicht nötig ist (z.B. wenn man eine Zahl nur einmal verwendet, braucht man dafür wirklich keinen Namen)

E8:T3 "Recursive Function Analysis"

- seid spezifischer. nicht "es" sondern n ist streng monoton fallend, weil Es wäre doof, deswegen Punkte liegen zu lassen
- vergesst nicht den Base Case (meist n == 0) zu nennen

Kommentare zu [code] expert

E8:T2 "Decode Binary NZZ Front Page"

- const std::string&
- führt keine Variablen ein, wenn es nicht nötig ist (z.B. wenn man eine Zahl nur einmal verwendet, braucht man dafür wirklich keinen Namen)

E8:T3 "Recursive Function Analysis"

- seid spezifischer. nicht "es" sondern n ist streng monoton fallend, weil Es wäre doof, deswegen Punkte liegen zu lassen
- vergesst nicht den Base Case (meist n == 0) zu nennen
- benutzt ruhig mathematische Notation in den PRE/POSTs. Das ist meistens klarer als Worte

Fragen zu [code] expert eurerseits?

Lernziele

Intro

- □ class und struct definieren und mit ihnen umgehen können
- ☐ Operatoren *overloaden* können

Self-Assessment III

Self-Assessment III

☐ Als Hausaufgabe (via moodle)

Objekte

struct **VS** class

Unterschied?

struct **VS** class

Unterschied?

Der einzige Unterschied liegt in der default visibility.

Object Default Visibility struct public ("visible") class private ("invisible")

Man kann die visibility der Members bei beiden ändern, mit den keywords private: bzw. public: .

Unterschied?

Der einzige Unterschied liegt in der default visibility.

Object Default Visibility struct public ("visible") class private ("invisible")

Man kann die visibility der Members bei beiden ändern, mit den keywords private: bzw. public:.

Wann sollte man was verwenden?

struct **VS** class

Unterschied?

Der einzige Unterschied liegt in der default visibility.

Object Default Visibility struct public ("visible") class private ("invisible")

Man kann die visibility der Members bei beiden ändern, mit den keywords private: bzw. public: .

Wann sollte man was verwenden?

Eigentlich egal; Hauptsache die visibility stimmt.

Empfehlung: class für komplizierteres Zeugs das hauptsächtlich viele Memberfunktionen hat und struct für Datenbündel (Punkte, Personen, mathematische Objekte)

Fragen/Unklarheiten?

"Wie weiss der Computer, welche Funktion man aufrufen möchte, wenn mehrere Funktionen denselben Namen haben?"

"Wie weiss der Computer, welche Funktion man aufrufen möchte, wenn mehrere Funktionen denselben Namen haben?"

 \rightarrow Woran kann der Compiler gleichnamige Funktionen unterscheiden?

"Wie weiss der Computer, welche Funktion man aufrufen möchte, wenn mehrere Funktionen denselben Namen haben?"

 \rightarrow Woran kann der Compiler gleichnamige Funktionen unterscheiden?

Kriterien

"Wie weiss der Computer, welche Funktion man aufrufen möchte, wenn mehrere Funktionen denselben Namen haben?"

ightarrow Woran kann der Compiler gleichnamige Funktionen unterscheiden?

Kriterien

- Anzahl der Funktionsparameter
- Typ der Funktionsparameter

keine Kriterien

"Wie weiss der Computer, welche Funktion man aufrufen möchte, wenn mehrere Funktionen denselben Namen haben?"

ightarrow Woran kann der Compiler gleichnamige Funktionen unterscheiden?

Kriterien

- Anzahl der Funktionsparameter
- Typ der Funktionsparameter

keine Kriterien

- Name der Funktionsparameter
- Return-Typ der Funktion

Was juckt uns das, dass zwei Funktionen den gleichen Namen haben können?

Overloading 000000

Function Overloading

Was juckt uns das, dass zwei Funktionen den gleichen Namen haben können?

→ Weil wir so *Operator Overloading* viel besser verwenden können!

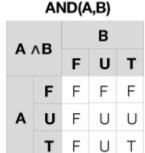
Was juckt uns das, dass zwei Funktionen den gleichen Namen haben können?

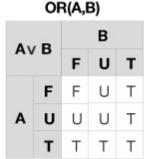
→ Weil wir so *Operator Overloading* viel besser verwenden können!

Operatoren (*, +, =, uvm.¹) sind eigentlich nur fancy Funktionen. Der Operator * beispielsweise hat mehrere Bedeutungen. Die richtige Deutung erhält man durch die Inputs. So kann man eine Art Multiplikation mit * einführen, die für std::vector<double> etwas anderes macht als für int.

Tribool: ein bool, aber mit drei möglichen Werten: false, unknown, und true.







F = FALSE, U = UNKNOWN, T = TRUE

Diese Implementation speichert die Wahrheitswerte in einer class namens Tribool und zwar als private unsigned int.

Diese Implementation speichert die Wahrheitswerte in einer class namens Tribool und zwar als private unsigned int.

- Wieso ist das eine gute Idee?
- Wie könnte man den Wahrheitswert speichern?

Diese Implementation speichert die Wahrheitswerte in einer class namens Tribool und zwar als private unsigned int.

- Wieso ist das eine gute Idee?
- Wie könnte man den Wahrheitswert speichern?
- Diese Aufgabe gehen wir zusammen an

Aufgabe "Tribool" Konzepte

Folgenden Konzepten und Keywords werden wir beim Lösen dieser Aufgabe begegnen:

- Classes und Structs
- Visibility
- Operator Overloading
- Deklaration vs Definition
- Out-of-Class-Definitionen
- const -Funktionen
- Konstruktoren ("C-tors")
- Member Initializer Lists
- **.** . . .

■ Step 1: Den ersten Konstruktor implementieren wir zusammen. Den zweiten versucht ihr selbst zu implementieren

- **Step 1:** Den ersten Konstruktor implementieren wir zusammen. Den zweiten versucht ihr selbst zu implementieren
- Step 2: Versucht Step 2 ebenfalls selbst

- Step 1: Den ersten Konstruktor implementieren wir zusammen. Den zweiten versucht ihr selbst zu implementieren
- Step 2: Versucht Step 2 ebenfalls selbst
- Step 3: Zuerst kurze Diskussion zu einer kleveren Implementationsmöglichkeit und dann könnt ihr es selbst versuchen.

Bis zum nächsten Mal

- ☐ Self-Assessment III
- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund

Allgemeine Fragen?