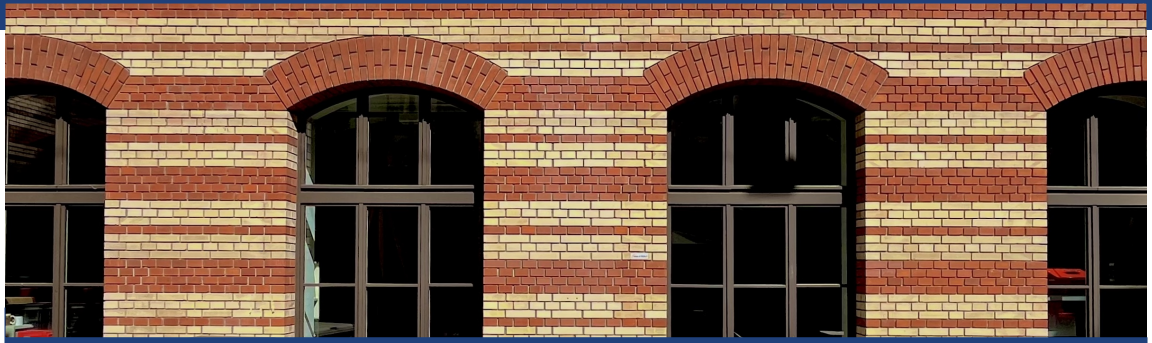


Übungsstunde — Informatik — 05

Adel Gavranović

code expert -Änderungen, assert, PRE und POST, Funktionen, Headerfiles, Namespaces

Übersicht

code expert Neuerungen
`assert`
PRE und POST
Functions
Exam Question
Headers und Namespaces
Stepwise Refinement
Alte Prüfungsfragen

n.ethz.ch/~agavranovic

Material
Webpage
Mail

1. Intro

Intro

- Hoffe euch hat die Session mit Ioana[1] gefallen! :)

[1] Ioana's Webpage

2. Follow-up

Follow-up aus letzter Übungsstunde

- Gab es noch offene Fragen bezüglich letzer Übungsstunde?
- Who liked it better in english?

3. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

- Ihr müsst keine <> setzen beim beantworten der Fragen. Schaut euch das kompilierte markdown-file einfach beim schreiben an :)
- Bitte löscht die Aufgabenbeschreibung, Tips und jegliche TODOs aus euren Abgaben (sonst gibt's Punkteabzug) ausser ihr macht was sinnvolles mit ihnen
- Falls jemand denkt, einen guten Grund (z.B. Militärdienst) zu haben, wieso eine verspätete Abgabe noch gewertet werden soll, so soll die Person mir mit kontaktieren (am besten via Mail)

Informationen bezüglich Bonusaufgabe I

- 10 Reguläre Tests ($\approx 71\%$)
- 4 Versteckte Tests
- Keine TA-Punkte von mir (aber wenn ihr wollt, dass ich den Code verstehe, gebt euch beim Formatting Mühe)
- Exericses ab "Perpetual Calendar" sammeln Punkte für die Bonusaufgabe II

Allgemeines bezüglich **code** expert

- Sind alle mit der Art des Feedbacks zufrieden?
- Gibt es Worte/Ausdrücke im Feedback die ihr nicht versteht?
- Was könnte ich besser machen?

Fragen bezüglich **code** expert eurerseits?

4. Lernziele

Ziele für Heute

Lernziele

- ☐ in der Lage sein, Vor- und Nachbedingungen (PRE and POST conditions) für Funktionen zu schreiben
- ☐ in der Lage sein, Aufgaben mittels "stepwise refinement" zu lösen
- ☐ in der Lage sein, `assert()` richtig anzuwenden
- ☐ in der Lage sein, Code, der Funktionen aufruft, zu tracen[2]
- ☐ Verstehen, was die Gefahren von `namespace` sind
- ☐ in der Lage sein, verständlichen und schönen Code zu schreiben[3]

[2] Program Tracing Guide

[3] Style Guide

5. Zusammenfassung

6. **code** expert Neuerungen

Neues Features: Versteckte Testfälle

Versteckte Testfälle

- Ab Woche 7 enthalten einige [code]expert-Übungen versteckte Testfälle
- Versteckte Testfälle zeigen bei Fehlern die erwartete Ausgabe nicht an, um Hard-Coding[4] zu verhindern
- Versteckte Testfälle sind in Bonusübungen und Prüfungen üblich
- Übungen mit versteckten Testfällen und persistenter Eingabe werden in **code** expert mit "[hidden tests]" gekennzeichnet

[4] Ohnehin verboten

Neues Features: Persistente Eingabe

Persistente Eingabe

- Neues Feature eingeführt, um Eingaben (aus `input.txt`) über mehrere Durchläufe zu speichern und wiederzuverwenden
- Die persistente Eingabe hat keinen Einfluss auf Noten oder XP
- Nutzung: `f` (ohne Leerzeichen) statt eigentliche Eingabe in Terminal eingeben[5]

[5] Eselsbrücke: `f` wie file!

Fragen/Unklarheiten?

7. assert

assert

- Wir werden Exit-Codes sehen, mehr Infos hier[6]

[6] Exit Code Guide

`assert`

Frage

- Wofür sind asserts nützlich?

Mögliche Antworten

- Um herauszufinden, wo genau ein Fehler passiert
- In langen Programmen, um den Überblick zu bewahren
- Falsche (User) Inputs sofort erkennen (hilft, *undefined behavior* zu vermeiden)
- Als eine Art der Code Dokumentation

`assert` Demo

code expert Code Example "Debugging with Assert"

Fragen/Unklarheiten?

8. PRE und POST

PRE und POST Conditions

```
// PRE:  describes accepted input
// POST: describes expected output
int yourfunction(int a, int b){
    ...
}
```

PRE und POST Conditions

Frage: Was wären hier sinnvolle Conditions?

```
// PRE:
// POST:
double area(double height, double length){
    return height*length;
}
```

Sie müssen nicht sehr detailiert sein, sie sollten beschreiben, was die Funktion erwartet und was sie ausgeben wird (wenn der Input erwartungsgemäss war)

Fragen/Unklarheiten?

PRE und POST Conditions I

Bestimme sinnvolle PRE- und POST-conditions für die folgende Funktion

```
// PRE:  ???
// POST: ???
double f(double i, double j, double k){
    if(i > j){
        if(i > k){return i;}
        else {return k;}
    } else {
        if(j > k){return j;}
        else {return k;}
    }
}
```

PRE und POST Conditions I (Lösung)

Mögliche Lösung

```
// PRE:  (not needed)
// POST: return value is maximum of {i, j, k}
double f(double i, double j, double k){
    if(i > j){
        if(i > k){return i;}
        else {return k;}
    } else {
        if(j > k){return j;}
        else {return k;}
    }
}
```

PRE und POST Conditions II

Bestimme sinnvolle PRE- und POST-conditions für die folgende Funktion

```
// PRE:  ???
// POST: ???
double g(int i, int j){
    double r = 0.0;
    for(int k = i; k <= j; k++){
        r += 1.0 / k;
    }
    return r;
}
```

PRE und POST Conditions II (Lösung)

Mögliche Lösung

```
// PRE:  0 not in [i, j] and i <= j < INT_MAX
// POST: return value is the sum 1/i + 1/(i+1) + ... + 1/j
double g(int i, int j){
    double r = 0.0;
    for(int k = i; k <= j; k++){
        r += 1.0 / k;
    }
    return r;
}
```

9. Functions

Output?

```
int f(int i){
    return i * i;
}

int g(int i){
    return i * f(i) * f(f(i));
}

int h(int i){
    std::cout << g(i) << "\n";
}
// ...
```

```
// ...
int main(){
    int i;
    std::cin >> i;
    h(i);
    return 0;
}
```

Was wird (mögliche Over- und Underflows vernachlässigt) der Output sein? **Lösung**: i^7

Fehlersuche

```
double f(double x){
    return g(2.0 * x);
}

double g(double x){
    return x % 2.0 == 0;
}

double h(double x){
    std::cout << result;
}
// ...
```

```
// ...
int main(){
    double result = f(3.0);
    h();

    return 0;
}
```

Finde mindestens 3 Fehler in diesem Programm.

Fehlersuche (Lösung)

1. `g()` ist `f()` nicht bekannt, da der Scope von `g()` erst später anfängt
2. Es gibt keinen `%`-Operator für `double`
3. `h()` "sieht" die Variabel `result` nicht, da sie nicht im gleichen Scope ist
4. Die Funktion `h()` hat keine Rückgabe, obwohl es eine braucht
5. `h()` wird ohne Argument aufgerufen

Anzahl Divisoren

Schreibe eine Funktion `number_of_divisors()`, die `int n` als Argument entgegennimmt und die Anzahl Divisoren von n zurückgibt (inklusive 1 und n)

```
// PRE:  0 < n < MAX_INT
// POST: returns number of divisors of n (incl. 1 and n)
unsigned int number_of_divisors(int n){
    // ...
}
```

Beispiel
6 hat 4 Divisoren, nämlich $1, 2, 3, 6$

Anzahl Divisoren (Lösung)

```
// PRE:  0 < n < MAX_INT
// POST: returns number of divisors of n (incl. 1 and n)
unsigned int number_of_divisors(int n){

    assert(n > 0);
    unsigned int counter = 0;

    for (int i = 1; i <= n; ++i){
        if(n % i == 0){
            counter++;
        }
    }

    return counter;
}
```

Fragen/Unklarheiten?

10. Exam Question

Prüfungsrelevant?

- Das ist eine echte Prüfungsaufgabe aus dem Jahr 2022
- Öffnet die Aufgabe "[Exam 2022.02 (MAVT + ITET)] Decimal to arbitrary base" auf **code** expert
- Besprecht die Aufgabe und euren Lösungsansatz mit euren Nachbar:innen
- Löst die Aufgabe

Fragen/Unklarheiten?

11. Headers und Namespaces

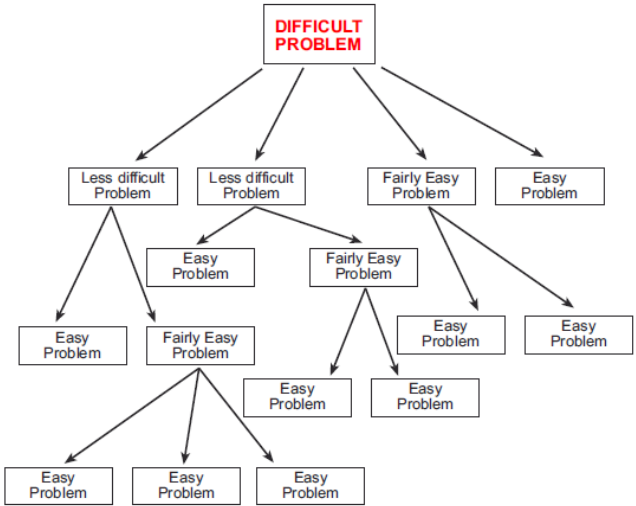
Headers und Namespaces

Live Demo Code Example "CPP Headers & Namespaces"

Fragen/Unklarheiten?

12. Stepwise Refinement

Grundidee


DIFFICULT PROBLEM
Less difficult Problem
Less difficult Problem
Fairly Easy Problem
Easy Problem
Easy Problem
Fairly Easy Problem
Easy Problem
Fairly Easy Problem
Easy Problem
Easy Problem
Easy Problem
Easy Problem
Easy Problem
Easy Problem
Easy Problem

Stepwise Refinement

Code Example "Perfect Numbers" on code expert

Write a program that counts how many perfect numbers exist in the range $[a, b]$. Please use stepwise refinement to develop a solution to this task that is divided into meaningful functions. We provide a function `is_perfect` in `perfect.h` that checks if a given number is perfect.

A number $n \in \mathbb{N}$ is called perfect if and only if it is equal to the sum of its proper divisors. For example:

- $28 = 1 + 2 + 4 + 7 + 14$ is perfect
- $12 \neq 1 + 2 + 3 + 4 + 6$ is not perfect

Stepwise Refinement

- *nicht sofort programmieren!*
- identifiziert die einfacheren Teilprobleme
- welche Teilprobleme konntet ihr identifizieren?

"Problembaum"

Wie viele vollkommene Zahlen (engl. perfect numbers) gibt es in $[a, b]$?

Lösung zu "Perfect Numbers"

```
// PRE:
// POST:
bool is_perfect(unsigned int number) {
  unsigned int sum = 0;
  for (unsigned int d = 1; d < number; ++d) {
    if (number % d == 0) {
      sum += d;
    }
  }
  return sum == number;
}
```

Lösung zu "Perfect Numbers"

```
#include <iostream>
#include "perfect.h"

// PRE:
// POST:
unsigned int count_perfect_numbers(unsigned int a, unsigned int b) {
  unsigned int count = 0;
  for (unsigned int i = a; i <= b; ++i) {
    if (is_perfect(i)) {
      count++;
    }
  }
  return count;
}

// ...
```

Lösung zu "Perfect Numbers"

```
// ...

int main () {
  // input
  unsigned int a;
  unsigned int b;
  std::cin >> a >> b;

  // computation
  unsigned int count = count_perfect_numbers(a, b);

  // output
  std::cout << count << std::endl;

  return 0;
}
```

Fragen/Unklarheiten?

13. Alte Prüfungsfragen

Prüfungsfrage "Typ und Wert"

Geben Sie den Typ und den Wert der Variablen `c` an[7]

```
int a = 5;
int b = 1;
auto c = (9 * a + b) % a;
```

Lösung

`int`, `1`

```
int a = 5;
double b = 1;
auto c = (9.0 * a + b) / a;
```

Lösung

`double`, `9.2`

[7] Bemerkung zu Typ- und Wertfragen: Das Keyword **`auto`** bedeutet, dass der Typ des Ausdrucks durch den Compiler bestimmt wird. Im Folgenden steht es also für den Typ des Ausdrucks, die Sie identifizieren müssen.

Prüfungsfrage F^*

Sei F^* das folgende normalisierte Fliesskommazahlensystem[8]

$$F^*(\beta = 2, p = 3, e_{\text{min}} = -1, e_{\text{max}} = 4)$$

Richtig oder Falsch?

1. "1.25 kann im Fliesskommazahlensystem exakt dargestellt werden"
   Richtig, nämlich $1.01 \cdot 2^0$
2. "Es existiert keine Zahl $Z \in F^*$, für die gilt: $0.0625 < Z < 0.25$"
   Richtig, die kleinste darstellbare Zahl ist 0.5 (i.e., $1.0 * 2^{-1}$)
3. "3.25 kann genau in F^* dargestellt werden"
   Falsch, $3.25 = 1,101 * 2^1$ würde die Genauigkeit $p \geq 4$ erfordern

[8] Erinnerung: die Genauigkeit (Anzahl der Ziffern) schliesst das führende Bit ein.

Prüfungsfrage "Schleife"

```
int sum = 17;
int i = 1;

do {
    i += sum;
    sum = sum / 2;
} while (i > sum && sum >= 0);

std::cout << sum;
```

Welche Aussage beschreibt den Output am besten?

- 17
- 8
- Terminiert nie
- 18

Prüfungsfrage "Loop Termination"

```
int sum = 17;
int i = 1;

do {
    i += sum;
    sum = sum / 2;
} while (i > sum && sum >= 0);

std::cout << sum;
```

Antwort: Es terminiert nie!

- Division von zwei positiven `int` kann nicht negativ sein
  $\Rightarrow$ `sum >= 0` ist immer wahr!
- Nach der ersten Ausführung des do-Blocks: `i > sum`.
  `sum` ist monoton fallend, `i` ist monoton steigend
  $\Rightarrow$ `i > sum` ist immer wahr.

14. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!