Übungsstunde

Woche 06

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Self-Assessment

PRE und POST

Funktionen

Stepwise Refinement

Outro

Prüfungsfrage

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Follow-up aus vorheriger Übungsstunde

- s.22 in den annotierten Slides zu NFPS wurde mit einem Kommentar ergänzt
- ”Trailing zeros” verändern die Zahl nicht, d.h., in einem NFPS F^* gilt (wie in der Mathematik):

$$1.1000_2 = 1.100000000000000_2 = 1.1_2$$

- Somit wäre die Zahl 1.100000_2 in einem F^* mit $p = 3$
- Frage: Wäre dann auch -1.100000_2 in diesem F^*?
- Antwort: Ja

Kommentare zu [code]expert

- Tut mir Leid für die späte Korrektur
- Wenn steht, ich hätte Feedback hinterlassen, ihr aber im Fenster wo normalerweise das Feedback ist, nichts seht, ist das kein Bug. Ich muss ” ” eingeben, damit ich ein Feedback abschicken kann
- Kein Feedback $\iff$ Nichts zu bemängeln $\iff$ Sehr gut!
- Können alle meine Korrekturen des Codes einsehen?
- Englischsprachiges Feedback bei allen okay?
- Verständlichkeit bei allen okay? (Maske, Nuscheln, Deutsch)
- Bitte sagt mir, wenn ihr findet, meine Feedbacks seien zu harsch oder zu kritisch
- Auch kurze Antworten sind richtig
- Ihr dürft die Zusammenfassungen für das Lösen der Aufgaben verwenden!
- **Bonus Exercise:** Die ”hidden tests” werden zwar edge cases testen, aber alles wir ”reasonable Input” sein

Kommentare E3:T4b ”Fibonacci overflow”

- Gebt nicht zu früh ab, ihr kennt euer Feedback von letzter Woche ja eigentlich noch nicht
- Manchmal ist XP wichtiger als die eleganteste Lösung, also stellt sicher, dass ihr sicher mal etwas korrektes bei allen Tasks abgeben könnt und dann wenn ihr Zeit habt könnt ihr es noch eleganter gestalten

Kommentare E4:T1-3 ”Loop mix-up”

- Wenn ein Link zur Sandbox gegeben ist, erwarten wir schon, das der Code auch läuft
  - Viele haben Code abgegeben, der nicht läuft, weil `again` nicht an der richtigen Stelle initialisiert wurde
- Da es eigentlich keine korrekten Lösungen gibt für die jew. 2. Frage bei den Loop mix-up Aufgaben, schaut euch einfach noch die ML dazu an
- Bei ”What does the code do?” wollen wir eine grobe übersicht *was* der Code macht, nicht *wie* er etwas macht, d.h.:
  - ”Berechnet $a \bmod b$” und nicht
  - ”Zuerst wird der User Input in `a` und `b` gespeichert, dann wird wiederholt b von a abgezogen, und zwar in einer `while`-Schleife *bla bla bla*. . .”

Kommentare E4:T4 ”Loop Analysis”

Induktive Beweise in der Informatik:

1. ”Zu Beginn des Loops gilt X”
2. ”Wenn zu Beginn der Loops X gilt, dann gilt am Ende des Loops X'”
3. Dann folgt eine Aussage darüber was sich zwischen X und X' getan hat, bspw. ”n bleibt konstant, aber i ist *streng monoton steigend* bei jeder Iteration der Loop”
4. Abschluss: ”Da $i > 0$ und *streng monoton steigend* ist, und zu Beginn galt $n > i$, wird irgendwann $i = n$, was die Condition (`i < n`) verletzt, und somit die Loop *terminiert*

Magic Words: *non-negative and strictly in/decreasing*

Kommentare E4:T5a,b ”Approximation of π”

- Bitte gebt euren Variabeln gute Namen
  - nicht immer nur einzelne Buchstaben
  - am besten selbsterklärende Namen wie
    - `count, lenght, sum, pi, denominator`
  - haltet euch weitestgehend an die Konventionen aus der Mathematik ($\mathtt{x} \in \mathbb{R}$, $\mathtt{n} \in \mathbb{N}$, etc.)
- Variablen für Loops sollten *nicht* ganz oben initialisiert werden, sondern erst in der loop selbst

  `for(int i = 0; ...; i++)`
- Kommentare, die einfach in Worte fassen, was der Computer mit den Variablen macht, könnt ihr euch sparen. Erklärt, kurz und bündig, die Idee hinder dem Codeabschnitt und *was* er macht und nicht *wie*

Fragen zu [code]expert eurerseits?

Lernziele

- ☐ Gute PRE- und POST-Conditions schreiben können
- ☐ Aufgaben mittels *Stepwise Refinement* (dt. *Schrittweise Weiterentwicklung*) lösen können

Self-Assessment II

- Logt euch ins moodle ein und wartet
- Macht das Self-Assessment II (30 min Zeitlimit)
- Musterlösungen werden am Ende beim Review zu sehen sein
- Self-Assessments haben **keinen** Einfluss auf eure Endnote

Fragen/Unklarheiten?

PRE- und POST-Conditions

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

Frage

Was wären hier sinnvolle Conditions?

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

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?

Exercise 1

Exercise 1

Find **PRE- and POST-conditions** for this function.

1. Function:

```
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;
  }
}
```

Exercise 1

1. Function:

```
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-Condition:

(not needed)

POST-Condition:

```
// POST: return value is
//       the maximum of
//       i,j and k
```

Exercise 1

Find **PRE- and POST-conditions** for this function.

2. Function:

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

Exercise 1

2. Function:

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

```
PRE-Condition:    // PRE: 0 not contained in {i, ..., j} and i <= j
                  // and j < INT_MAX
POST-Condition:   // POST: return value is the sum
                  //         1/i + 1/(i+1) + ... + 1/j
```

Exercise 2

Exercise 2

- What is the **output** of this program?
- You can neglect possible over- or underflows for this exercise.

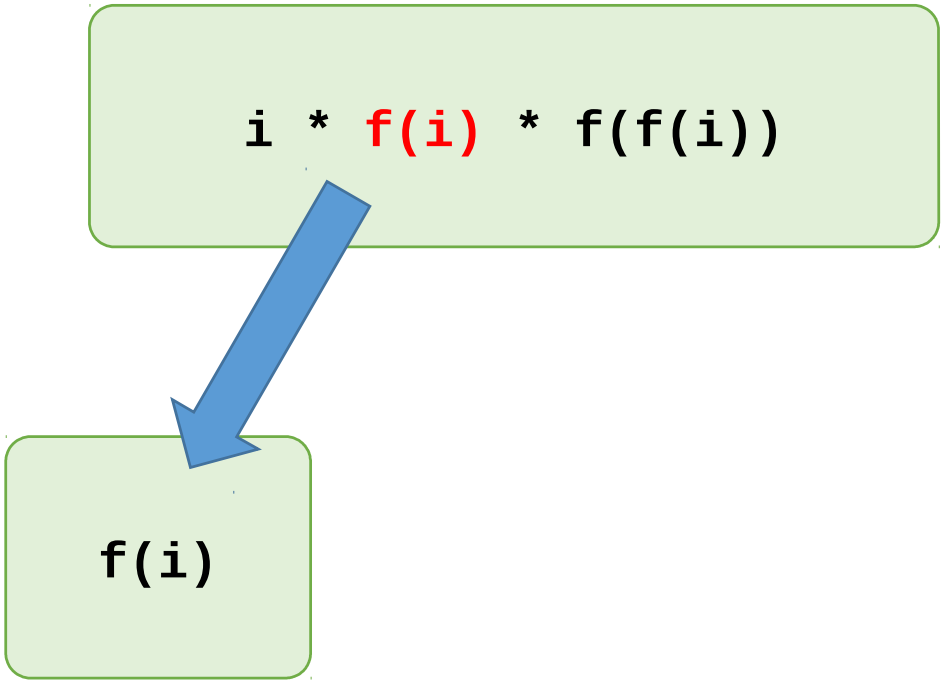
```
#include <iostream>

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

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

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

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

Exercise 2

```
i * f(i) * f(f(i))
```

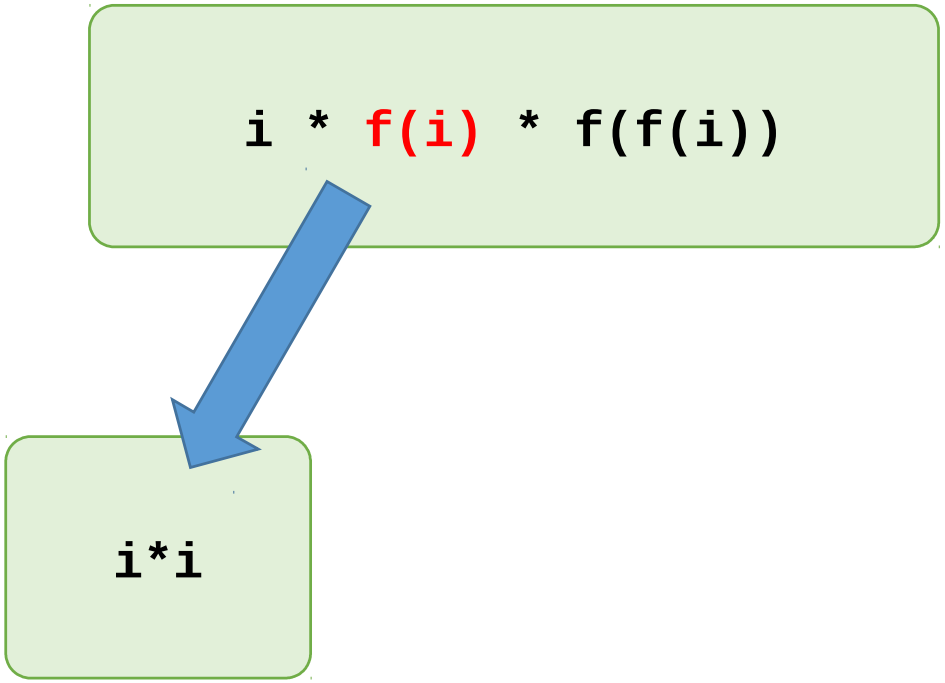
```
#include <iostream>

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

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

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

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

Exercise 2



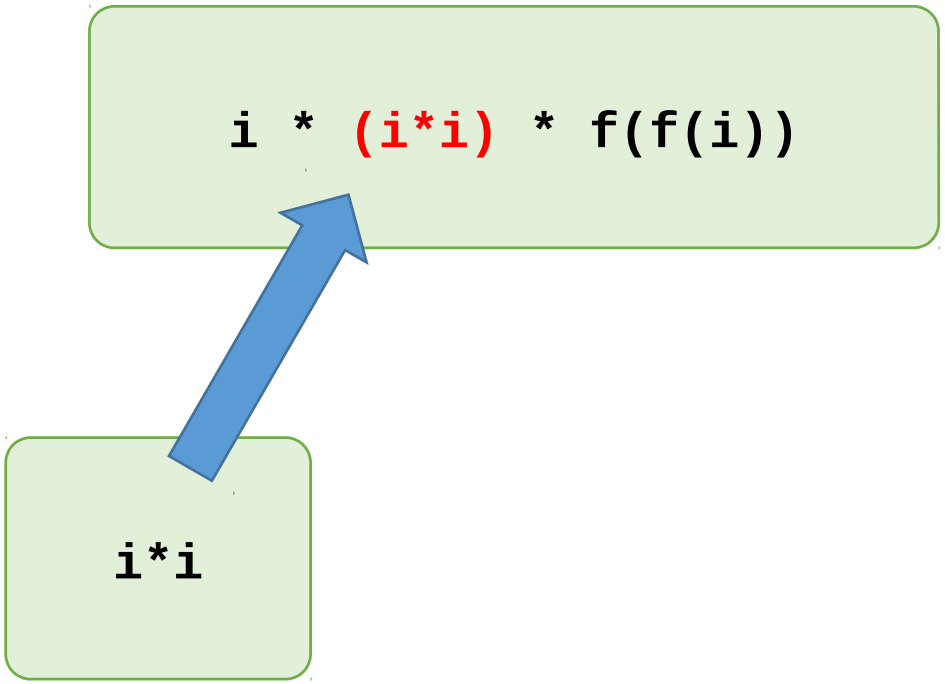

```
#include <iostream>

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

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

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

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

Exercise 2



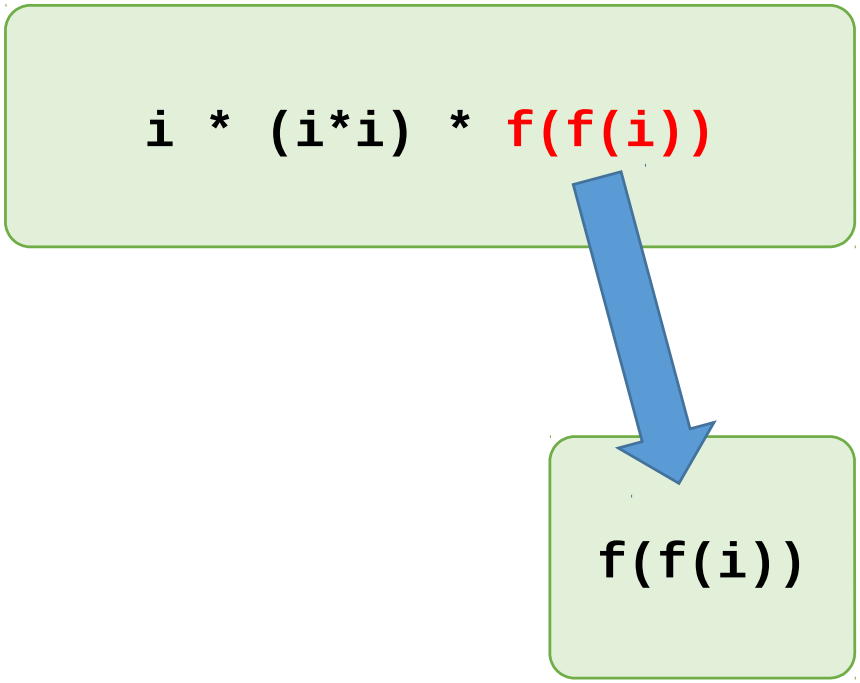

```
#include <iostream>

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

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

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

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

Exercise 2



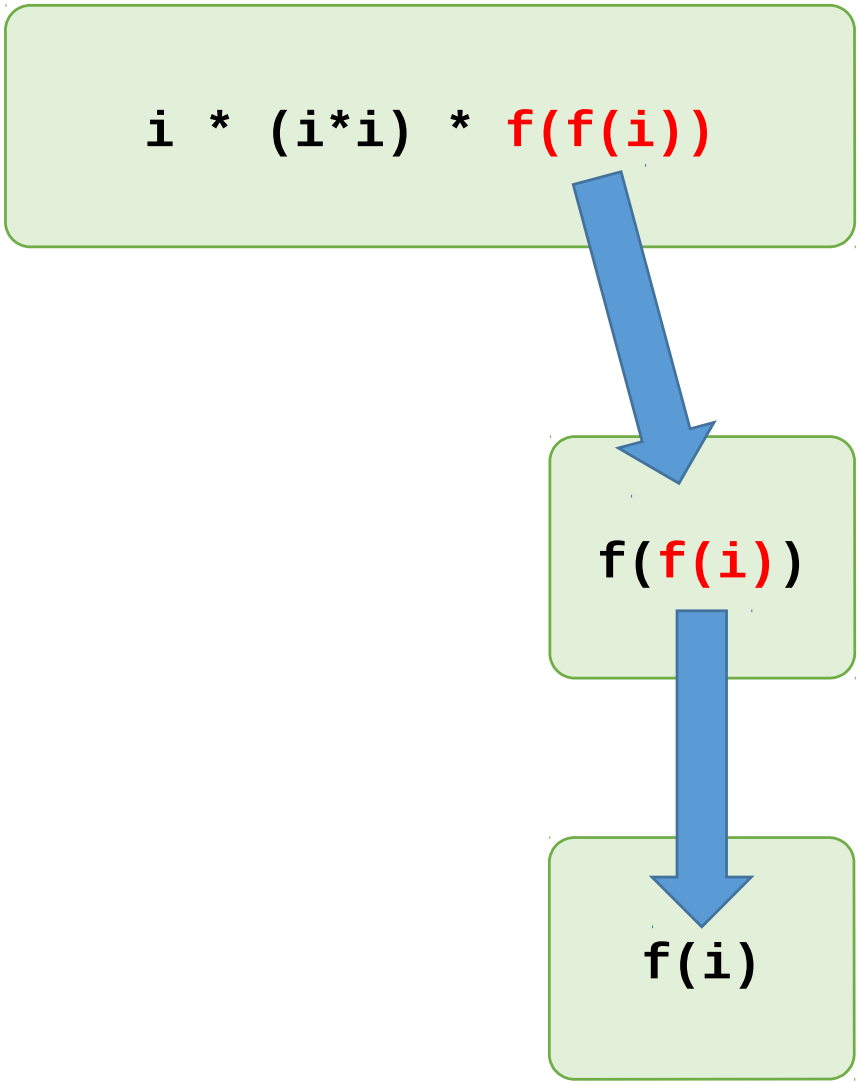

```
#include <iostream>

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

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

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

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

Exercise 2



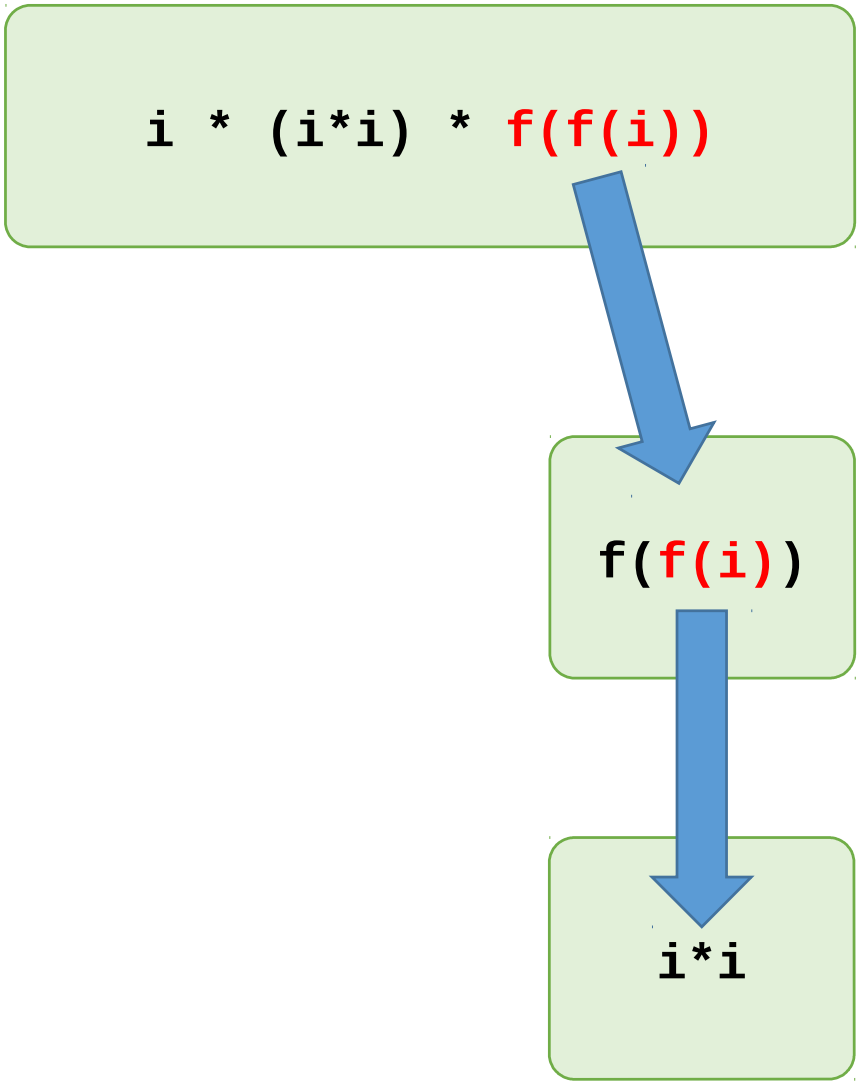

```
#include <iostream>

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

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

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

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

Exercise 2



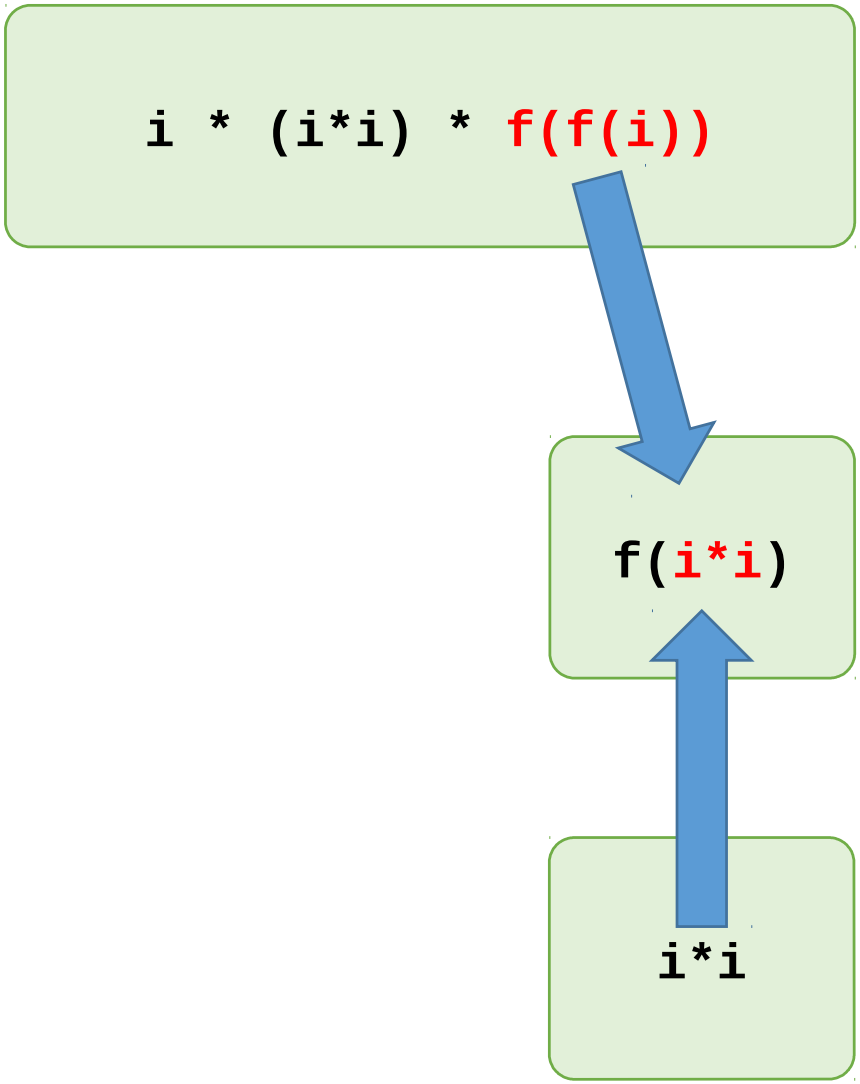

```
#include <iostream>

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

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

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

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

Exercise 2



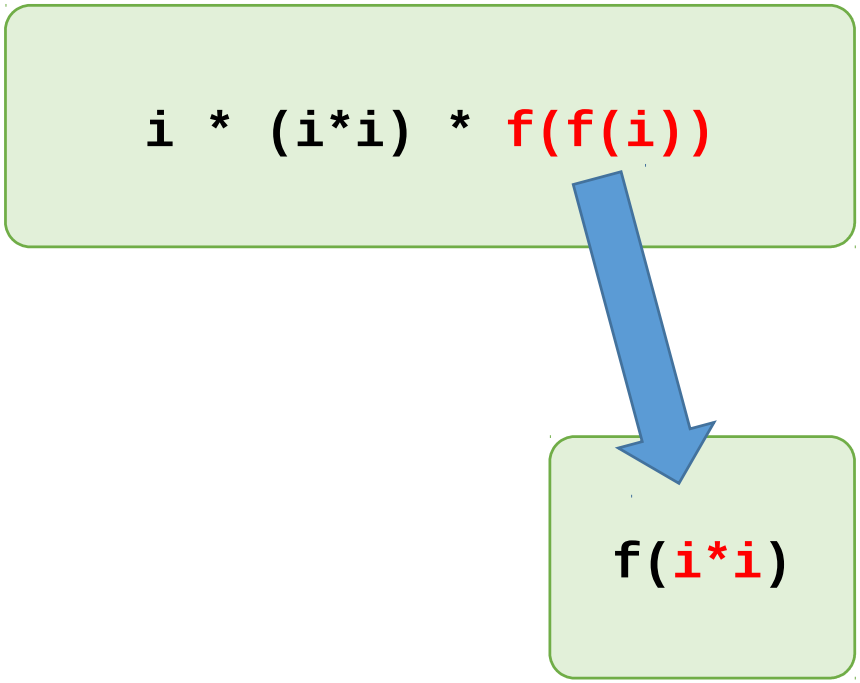

```
#include <iostream>

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

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

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

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

Exercise 2



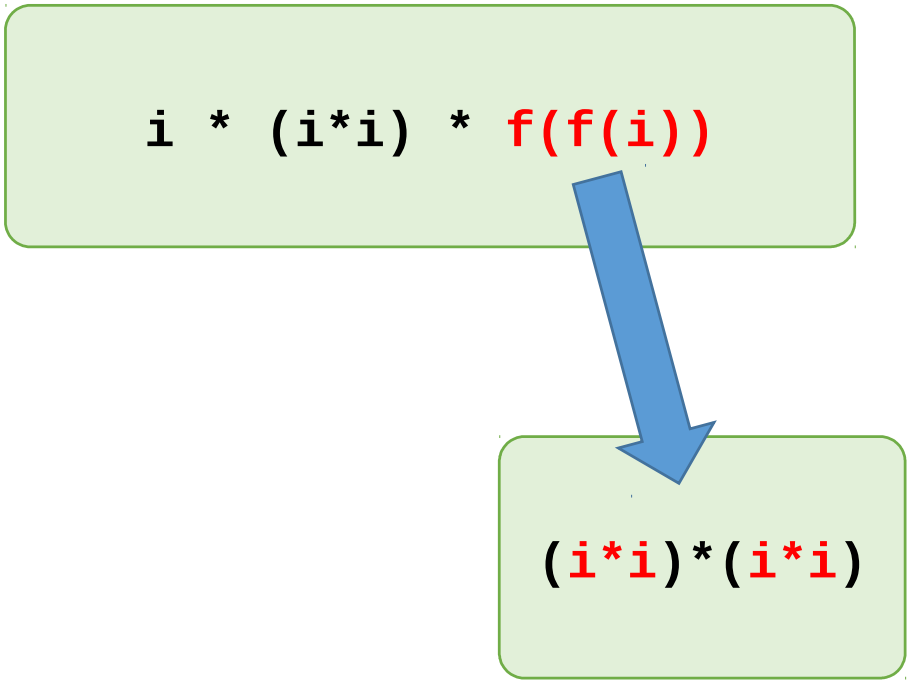

```
#include <iostream>

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

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

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

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

Exercise 2




```
#include <iostream>

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

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

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

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

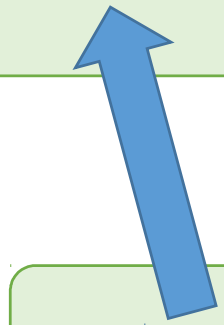
Exercise 2




```
#include <iostream>

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

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

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

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

Exercise 2

```
i * (i*i) * ((i*i)*(i*i))
```



```
(i*i)*(i*i)
```

```
#include <iostream>

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

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

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

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

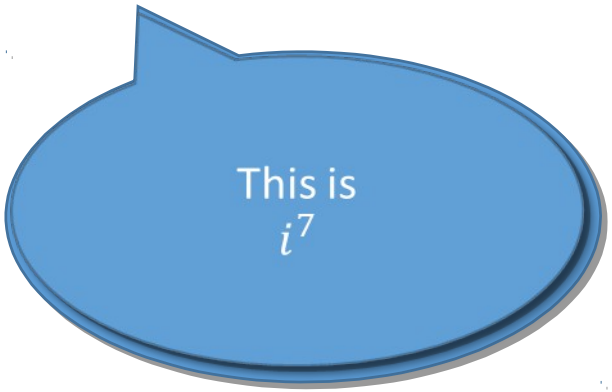
Exercise 2

```
i * (i*i) * ((i*i)*(i*i))
```

```
#include <iostream>

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

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

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

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

Exercise 2

```
i * (i*i) * ((i*i)*(i*i))
```




```
#include <iostream>

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

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

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

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

Exercise 1

Exercise 1

Find **3 mistakes** in this program.

```
# include <iostream>

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

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

void h () {
   std::cout << result;
}

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

   return 0;
}
```

(From: Script Exercise 82)

Exercise 1

Problem 1: g() not yet known

scope of g starts later

```
# include <iostream>

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

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

void h () {
   std::cout << result;
}

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

   return 0;
}
```

Exercise 1

Problem 1: `g()` not yet known

scope of `g` starts later

```
# include <iostream>

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

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

void h () {
   std::cout << result;
}

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

   return 0;
}
```

Problem 2: Modulo

no modulo for `double`

Exercise 1

Problem 1: `g()` not yet known

scope of `g` starts later

Problem 2: Modulo

no modulo for `double`

Problem 3: `h()` does not «see» `result`

`result` is out-of-scope

```
# include <iostream>

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

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

void h () {
   std::cout << result;
}

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

   return 0;
}
```

(From: Script Exercise 82)

Exercise 2

Exercise 2

Write a function `number_of_divisors` which takes an `int n` as argument and returns the number of divisors of `n` (including `1` and `n`).

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

Example:

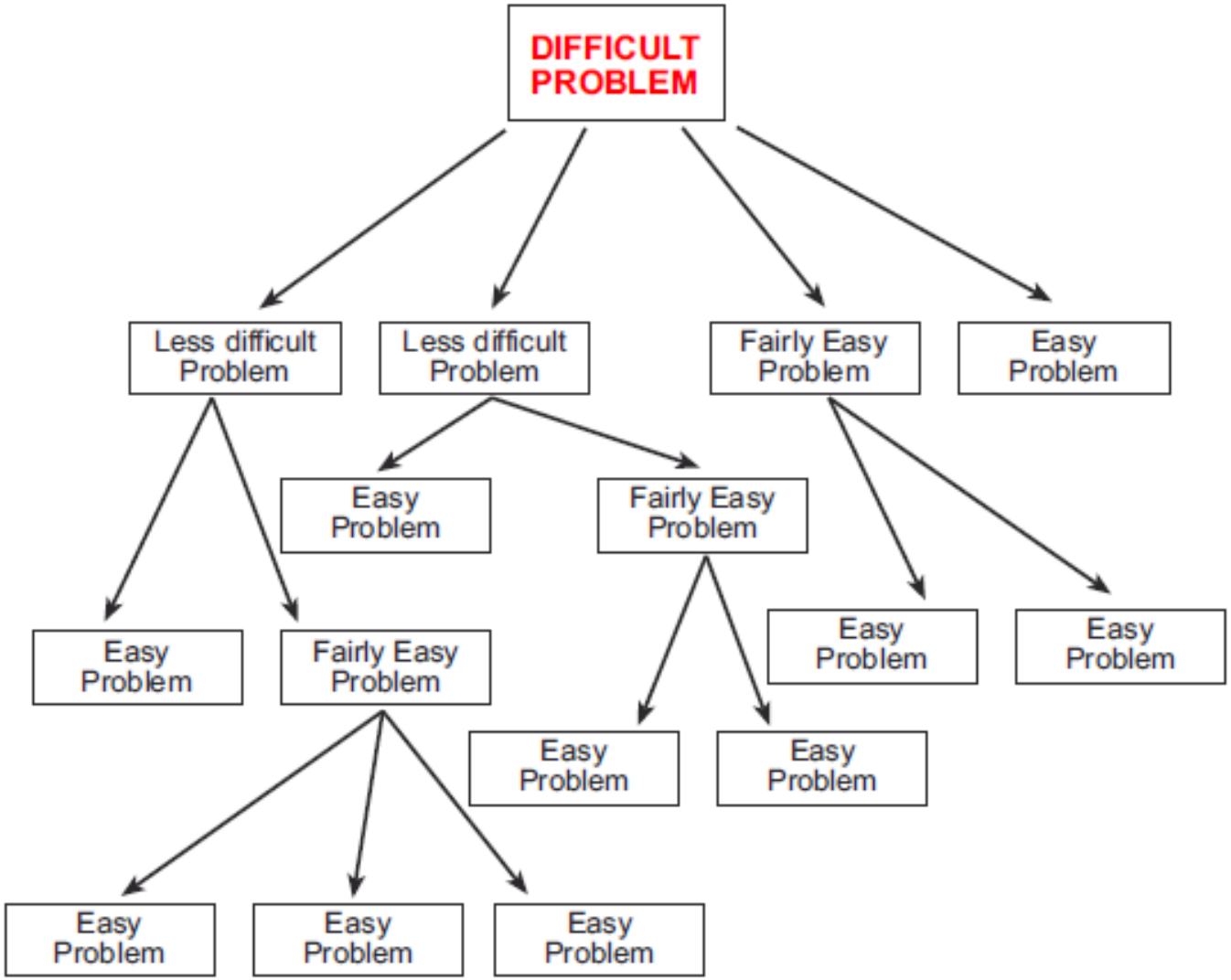
- 6 has 4 divisors, namely 1, 2, 3, 6
  → `std::cout << number_of_divisors(6); // output: 4`

Exercise 2

```
// PRE: n > 0
// 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?

Stepwise Refinement



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

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

”Problembaum”

Wie viele vollkommene Zahlen (engl. perfect numbers) gibt es?

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 and output
  unsigned int count = count_perfect_numbers(a, b);

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

  return 0;
}
```

Fragen/Unklarheiten?

Tipps für [code]expert

Bei allen ist Stepwise Refinement unabdingbar

E6:T1: "Perpetual calendar"

- fangt früh genug an
- arbeitet ruhig zusammen, aber kopiert nicht voneinander

E6:T2: "Run-length encoding"

- `break` darf benutzt werden
- gebt auch euren Funktionen auch selbsterklärende Namen (beispielsweise `is_byte(n)` um zu checken, ob $n \in [0, 255]$)
- schreibt so viel des Codes als Funktionen wie nur möglich

E6:T3: "Fixing Functions"

- Code-snippets für Antworten bitte richtig formatieren (wissen alle wie?)

Feedback an mich?

-
-
-
-
-

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ arbeitet vor, damit ihr Halloween geniessen könnt
- ☐ bleibt gesund und lasst euch nach den Parties testen

Allgemeine Fragen?

- insb. zum Vokabular?

Prüfungsfrage

EURE PRÜFUNG WIRD ANDERS - VERGESST DAS NICHT

(a) `4 <= 5.f`

Typ / *Type* | Wert / *Value*

(b) `2 / 4.0f + 7 / 2.0`

Typ / *Type* | Wert / *Value*

(c)
```
int a = 4;
int d = 5;
```

`a / d--`

Typ / *Type* | Wert / *Value*

(d) `std::vector<float> f = { 1, 2, 3, 4, };`

`f[2] + (int)f[3]`

Typ / *Type* | Wert / *Value*

(e)
```
int b = 3;
int k = b++;
```

`++k`

Typ / *Type* | Wert / *Value*

(f) `(2u - 8 % 5) < 0`

Typ / *Type* | Wert / *Value*

(a) `4 <= 5.f`

Typ / *Type*: `bool` | Wert / *Value*: `true`

(b) `2 / 4.0f + 7 / 2.0`

Typ / *Type*: `double` | Wert / *Value*: `4`

(c)
```
int a = 4;
int d = 5;
```

`a / d--`

Typ / *Type*: `int` | Wert / *Value*: `0`

(d) `std::vector<float> f = { 1, 2, 3, 4, };`

`f[2] + (int)f[3]`

Typ / *Type*: `float` | Wert / *Value*: `7`

(e)
```
int b = 3;
int k = b++;
```

`++k`

Typ / *Type*: `int` | Wert / *Value*: `4`

(f) `(2u - 8 % 5) < 0`

Typ / *Type*: `bool` | Wert / *Value*: `false`