ETH zürich

# Exercise Session W05

Computer Science (CSE) – AS 23

# Overview

**Today's Agenda**

Follow-up
Feedback on **code** expert
Objectives
PRE und POST
Funktionen
Exam Question
Stepwise Refinement
Outro

n.ethz.ch/~agavranovic

# 1. Follow-up

# Follow-up

- What is *Binary Expansion*?
  - This[1]
- When you ask me questions during the session, please make sure to also send me a follow-up e-mail so I know your name and can answer before the next session

[1] `https://lec.inf.ethz.ch/math/informatik_cse/2023/slides/lecture5.en.handout.pdf`

# 2. Feedback on **code** expert

# General things regarding **code** expert

- Please don't write past the gray line
  - Just use multi-line comments
- There's almost always a better approach; don't feel bad if you didn't get it the first time
- `n = n+1` and `n += 1` are not very idiomatic for C++, use `n++` instead
- The use ofMagic Numbers[2]must be explained
- Feel free to delete the `<Insert your answer here, within the comment block>` when answering questions

[2] `https://en.wikipedia.org/wiki/Magic_number_(programming)`

# General things regarding **code** expert

- Almost all the submissions were way wordier than needed
- What $\neq$ How
  - When asked *what* a code snippet does, don't explain *how* it does it
  - Hint: If you're mentioning variable names, you're probably not describing *what* something does but *how*
- If there were multiple similar exercises, extensive feedback was given to only one of them
- No feedback $\Longrightarrow$ Well done
- I can make changes/suggestions to your code and you're able to see it

# Questions?

# 3. Objectives

# Objectives

- ☐ Be able to write good PRE and POST conditions
- ☐ Be able to solve tasks using *Stepwise Refinement*.

# 4. PRE und POST

# PRE and POST Conditions

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

# PRE and POST Conditions

## Questions

What would be sensible conditions here?

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

They don't have to be very detailed but they have to describe what the function expects and what will be returned *if* the provided input matches the expectations

# Questions?

# 5. Funktionen

# 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

**PRE-Condition:**
(not needed)

**POST-Condition:**

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

## 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

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.

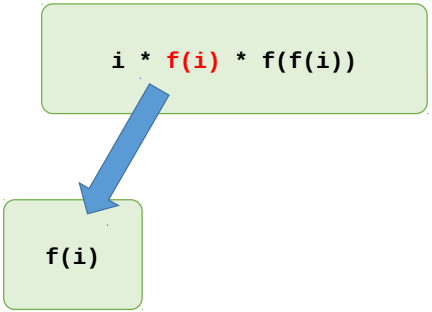
```
#include <iostream>

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))
```

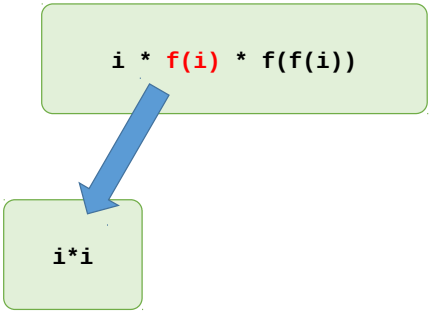
```
#include <iostream>

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



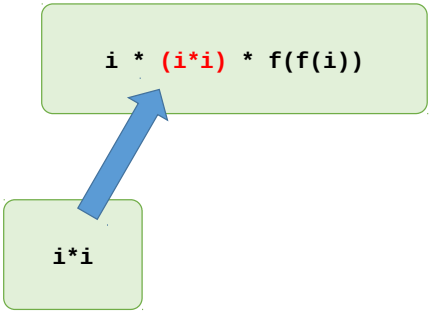

```
#include <iostream>

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



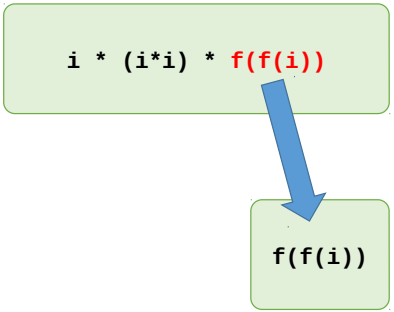

```
#include <iostream>

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



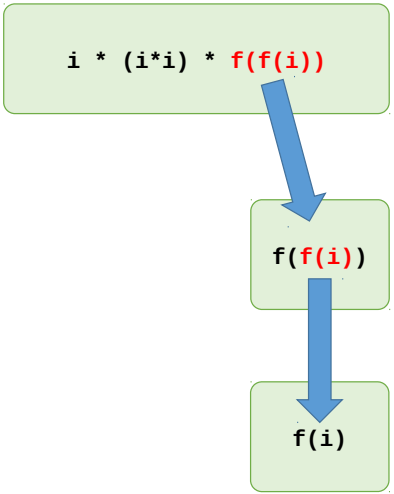

```
#include <iostream>

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



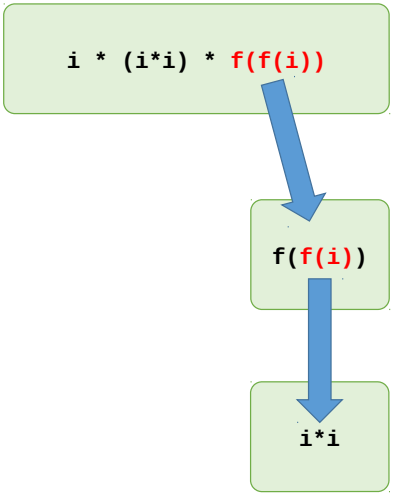

```
#include <iostream>

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



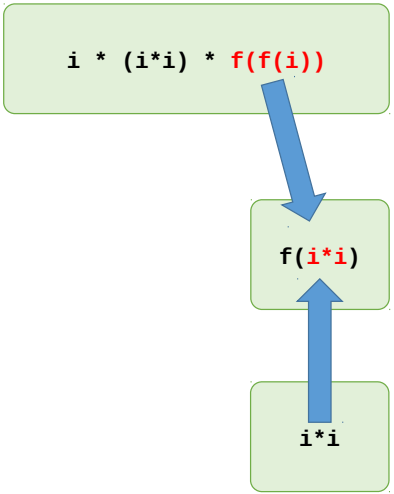

```
#include <iostream>

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



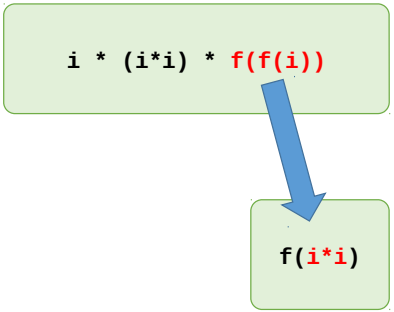

```
#include <iostream>

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



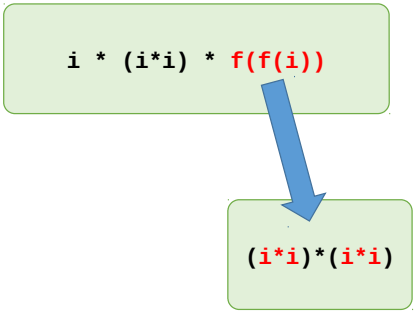

```
#include <iostream>

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



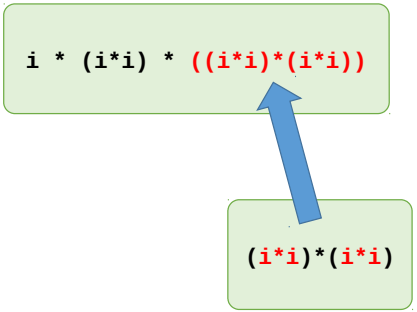

```
#include <iostream>

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




```
#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;
}
```

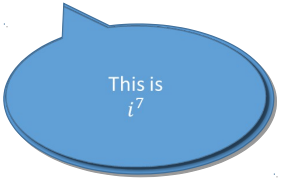
# Exercise 2

```
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;
}
```

(From: Script Exercise 82)

# Exercise 1

**Problem 1: g () not yet known**

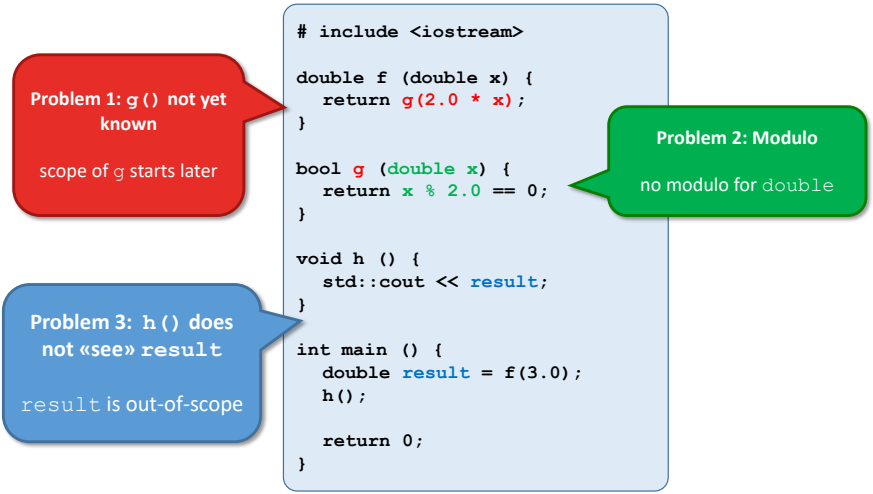
scope of g starts later

```
# 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;
}
```

**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;
}

# 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 and n < MAX_INT
// 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 and 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;
}
```

(Based on: Exam Summer 2012)

# Questions?

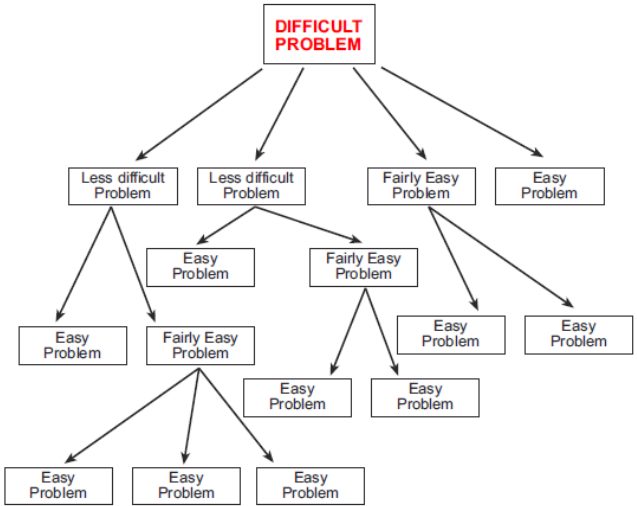
# 6. Exam Question

# Titel

- This is a real exam exercise from 2022
- Open the exercise “[Exam 2022.02 (MAVT + ITET)] Decimal to arbitrary base” on **code** expert
- Discuss you approach with your neighbours
- Solve the exercise

# 7. Stepwise Refinement

# 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

# Stepwise Refinement

- *Don’t start right away*
- Identify the easier subproblems
- What subproblems were you able to identify?

# “Problem Tree”

How many perfect numbers are there?

# Solution "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;
}
```

# Solution "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;
}

...
```

# Solution "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;
}
```

# Questions?

# 8. Outro

# General Questions?

# Till next time!

Cheers!