Exercise Session

Week 06

Adel Gavranović
agavranovic@student.ethz.ch

Overview

▸ polybox for session material ▸ Mail to TA

Today's Topics

Introduction

Self-Assessment

PRE and POST

Functions

Stepwise Refinement

First of all: Thanks!

Thank you all for the kind feedback!

I'll try to implement it in the future.

Unanswered Questions and Corrections from last Exercise Session

Note in advance

all of these questions are great and I love trying to answer them and learning new things myself, but please remember: very little (to none) of these questions *really* matter for the exam, so don't think you really have to know all the details. Some of these questions will cause you to go down a wikipedia rabbithole for hours — hours, which you could've spent studying and practising. But please don't ever lose you curiosity.

Unanswered Questions and Corrections from last Exercise Session

Do the floating-point-numbers we sum up have to be in the F^* already?

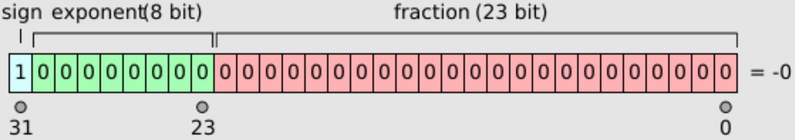
I couldn't find a satisfactory answer to this one. It *seems* like they would have to be inside it (much like in yesterday's exercises) because the program would first convert the given input into a NFP inside of F^* and then do arithmetic on it.

Unanswered Questions and Corrections from last Exercise Session

How *exactly* is 0 stored in a float?

Surprise! There are two 0’s! (and a few more special values we won’t cover)

sign = 0 for positive zero, 1 for negative zero
”biased” exponent = 0
fraction = 0



Unanswered Questions and Corrections from last Exercise Session

What if a number is *waaay* outside F^*?

Basically, the IEEE-754 tells us to just round it to the nearest number, in this case the greatest number in the set F^* **or** to set the ”number” to ∞. Which one of these options depends on what rounding is used. (Default: ∞)

Positive and negative infinity are represented thusly:
sign = 0 for positive infinity, 1 for negative infinity.
biased exponent = all 1 bits.
fraction = all 0 bits.

Comments on last [code]expert Exercises

- *Prove that the program terminates...*

Comments on last [code]expert Exercises

- *Prove that the program terminates...*
  - you usually have to show that for any (usually allowed) given input, the program (usually a loop) will somehow/at some point end. In many cases because a loop condition will turn false

Comments on last [code]expert Exercises

- *Prove that the program terminates...*
  - you usually have to show that for any (usually allowed) given input, the program (usually a loop) will somehow/at some point end. In many cases because a loop condition will turn false
  - always use the magic wording *”strictly monotonic in-/ or decreasing”*

Comments on last [code]expert Exercises

- *Prove that the program terminates...*
  - you usually have to show that for any (usually allowed) given input, the program (usually a loop) will somehow/at some point end. In many cases because a loop condition will turn false
  - always use the magic wording *”strictly monotonic in-/ or decreasing”*
  - *when* this happens is irrelevant, it just has to happen *at some point*, usually when `i = n` or similar

Comments on last [code]expert Exercises

- *Prove that the program terminates...*
  - you usually have to show that for any (usually allowed) given input, the program (usually a loop) will somehow/at some point end. In many cases because a loop condition will turn false
  - always use the magic wording *”strictly monotonic in-/ or decreasing”*
  - *when* this happens is irrelevant, it just has to happen *at some point*, usually when `i = n` or similar
  - possible trick question: something causes overflows and the loop/program goes on forever

Comments on last [code]expert Exercises

- *Prove that the program terminates...*
  - you usually have to show that for any (usually allowed) given input, the program (usually a loop) will somehow/at some point end. In many cases because a loop condition will turn false
  - always use the magic wording *"strictly monotonic in-/ or decreasing"*
  - *when* this happens is irrelevant, it just has to happen *at some point*, usually when `i = n` or similar
  - possible trick question: something causes overflows and the loop/program goes on forever
- Always try to turn a sum into one (not multiple) loops first

Question or Comments re: Exercises?

Learning Objectives Checklist

Now I...

□ can write PRE- and POST-conditions for simple functions

Learning Objectives Checklist

Now I...

- □ can write PRE- and POST-conditions for simple functions
- □ understand what stepwise refinement is

Learning Objectives Checklist

Now I...

- □ can write PRE- and POST-conditions for simple functions
- □ understand what stepwise refinement is
- □ can solve tasks using stepwise refinement

Self-Assessment

- Log into the Moodle page and wait

Self-Assessment

- Log into the Moodle page and wait
- Do the Self-Assessment (be aware of the 20 minute time limit)

Self-Assessment

$|10|.01_2$

$1.0_2 \cdot 2^1$

0: down
1 up

500

$\longrightarrow F^{*}(2, p = 2, \ldots)$

e_{max}

- Log into the Moodle page and wait
- Do the Self-Assessment (be aware of the 20 minute time limit)
- the Master Solution will be available when you review your solutions

Self-Assessment

- Log into the Moodle page and wait
- Do the Self-Assessment (be aware of the 20 minute time limit)
- the Master Solution will be available when you review your solutions
- this has **no** impact on your final grade

Self-Assessment

- Log into the Moodle page and wait
- Do the Self-Assessment (be aware of the 20 minute time limit)
- the Master Solution will be available when you review your solutions
- this has **no** impact on your final grade
- we'll discuss parts of it after you're done

Questions?

How to study for the exam?

- I would like to know if you already have a strategy
- Share your ideas and strategies with the group and get new ideas and feedback for yours (and I'll share mine at the end)

My ”study workflow” for Computer Science

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it
- **Practice!** try to do every exercise on `[code]`expert

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it
- **Practice!** try to do every exercise on [code]expert
- Note words and concepts you didn’t understand (fully) while solving the exercises or in class (ideally ask immediately and write it down). Go over these words/concepts at the end of the week and study them again and get help if needed

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it
- **Practice!** try to do every exercise on `[code]`expert
- Note words and concepts you didn’t understand (fully) while solving the exercises or in class (ideally ask immediately and write it down). Go over these words/concepts at the end of the week and study them again and get help if needed
- It’s *super* important to know ”what you don’t know yet”, hence the list of words/topics

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it
- **Practice!** try to do every exercise on [code]expert
- Note words and concepts you didn’t understand (fully) while solving the exercises or in class (ideally ask immediately and write it down). Go over these words/concepts at the end of the week and study them again and get help if needed
- It’s *super* important to know ”what you don’t know yet”, hence the list of words/topics
- Go over exercises you didn’t get right the first time periodically, to check and reevaluate your understanding of the topic/task. Pro tip: do this in the Lernphase too

My ”study workflow” for Computer Science

- Have a list of every topic covered in class and a way to indicate how well you understand it
- **Practice!** try to do every exercise on `[code]`expert
- Note words and concepts you didn’t understand (fully) while solving the exercises or in class (ideally ask immediately and write it down). Go over these words/concepts at the end of the week and study them again and get help if needed
- It’s *super* important to know ”what you don’t know yet”, hence the list of words/topics
- Go over exercises you didn’t get right the first time periodically, to check and reevaluate your understanding of the topic/task. Pro tip: do this in the Lernphase too
- You’ll feel dumb (often), but that’s okay. You’re here to make mistakes and learn

PRE and POST Conditions

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

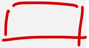
PRE and POST Conditions

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

Task

Write the PRE and POST conditions

```
// PRE:  H, L >= 0
// POST: Return  Area  of  H  and  L  box?
double A(double H, double L){
   return H*L;
}
```



(Note to self: use better variable names)

PRE and POST Conditions

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

Task

Write the PRE and POST conditions

```
// PRE:
// POST:
double A(double H, double L){
   return H*L;
}
```

They don't have to be extremely exact, but they should give you an idea of what the function expects and returns

Questions?

Functions

:: see function_exercises1.pdf ::

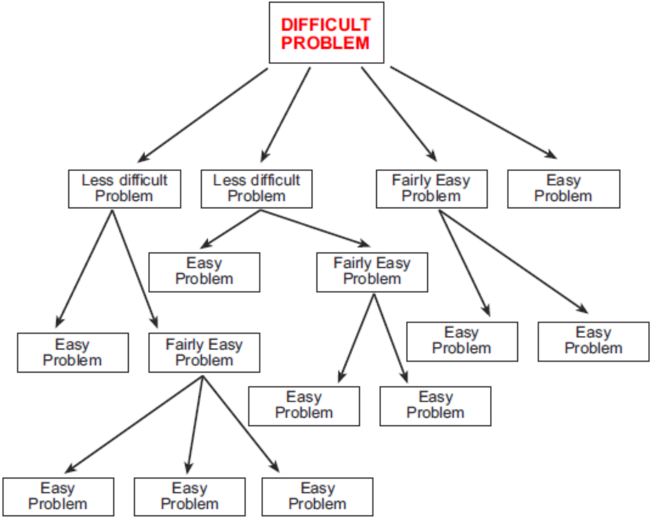
Functions

:: see function_exercises1.pdf ::

:: see function_exercises2.pdf ::

Questions?

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

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

- don’t try to solve it (yet)
- first identify the easier problems with pen and paper

Stepwise Refinement

Code Example ”Perfect Numbers” on [code]expert

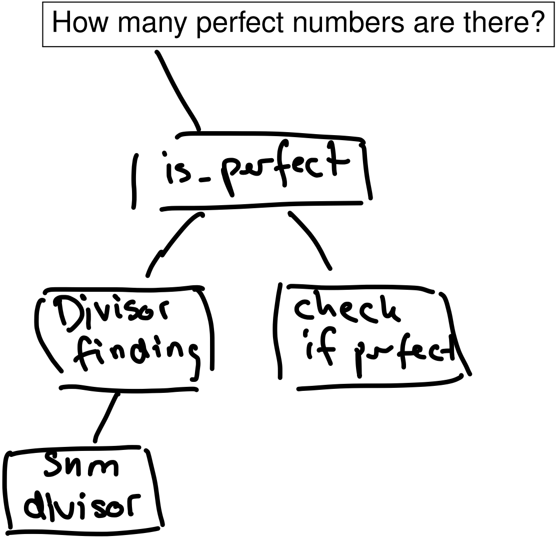
Write a program that counts how many perfect numbers exist in the range $[a, b]$. Please use stepwise refinement to develop a solution to this task that is divided into meaningful functions. We provide a function `is_perfect` in `perfect.h` that checks if a given number is perfect.

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

- don’t try to solve it (yet)
- first identify the easier problems with pen and paper
- share the problems you were able to identify

"Problem Breakdown Tree"

" diffucult"
Problem


How many perfect numbers are there?

Solution to ”Perfect Numbers”

```
// PRE: Positive number (int)
// POST: tell u if perfect or not (true,false)
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 to ”Perfect Numbers”

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

// PRE: a≤b, a≥0, b≥0
// POST: number of perf. number [a,b]
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 to ”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?