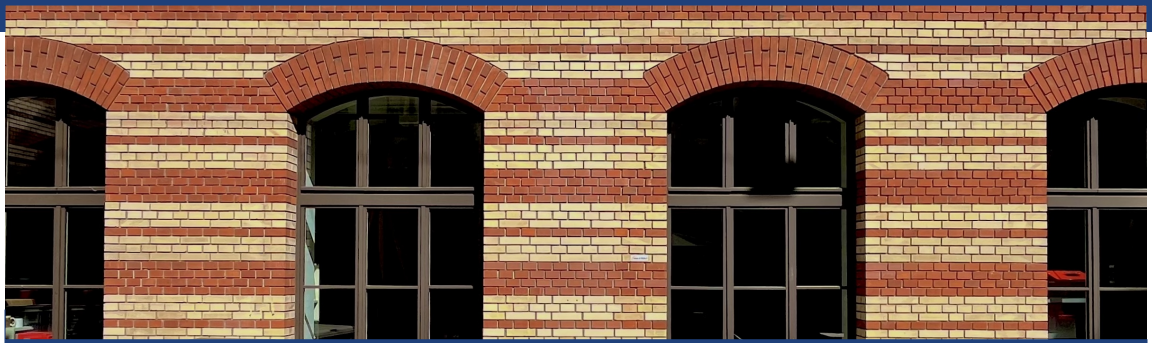


Exercise Session — Computer Science — 04

code expert Changes, assert, PRE and POST, Functions, Headers, Namespaces

Overview

Today's Plan

code expert Changes
`assert`
PRE and POST
Functions
Exam Question
Headers and Namespaces
Stepwise Refinement
Old Exam Question

1. Feedback regarding **code** expert

General things regarding code expert

- If someone thinks they have a good reason (e.g. military service) why a late submission should still be counted, they should contact me (preferably via email).
- Is everyone happy with the type of feedback I am providing?
- Is there something in the feedback you received that you don't understand?
- What could I do better?

Information regarding Bonus Exercise 1

- 10 Regular Tests (71%)
- 4 Hidden Tests
- No TA points from me (but if you want me to understand the code, make an effort with the formatting)
- Exercises from "Perpetual Calendar" onwards collect points for the Bonus Task II

Any questions regarding **code** expert on your part?

2. **code** expert Changes

New Feature: Hidden Test Cases

Hidden Test Cases

- Starting week 7, some **code** expert exercises will include hidden test cases
- Hidden test cases do not display expected output on failure to prevent hard-coding[1]
- Hidden test cases are common in bonus exercises and exams
- Exercises with hidden test cases and persistent input are labelled with "[hidden tests]" in **code** expert

[1] Forbidden anyway

New Feature: Persistent Input

Persistent Input

- Feature introduced to save and reuse input (from `input.txt`) across runs
- Persistent input does not affect grades or XP
- Usage: Enter `f` (without spaces) instead of the actual input in the terminal[2]

[2] Mnemonic: `f` like file!

Questions?

3. assert

Exit codes

- An exit code, also known as exit status, is an integer value that is returned by a program upon termination - see here for more information[3]
- Exit codes are used by the program's caller or user to determine whether the program's execution was successful.

[3] `https://lec.inf.ethz.ch/ifmp/2024/guides/debugging/exit_codes.html`

Exit codes

- An exit code, also known as exit status, is an integer value that is returned by a program upon termination - see here for more information[3]
- Exit codes are used by the program's caller or user to determine whether the program's execution was successful.
- Examples of exit codes:
  - 0 - Successful program termination
  - -6 - `assert(...)` failed
  - -8 - division by 0
  - -9 - program was killed because it used too much memory
  - -11 - segmentation fault

[3] `https://lec.inf.ethz.ch/ifmp/2024/guides/debugging/exit_codes.html`

assert

Question

- What do we use assert for?

assert

Question

- What do we use assert for?

Possible Answers

- To find out where exactly an error occurs
- To keep a better overview of long programs
- To catch wrong (user) inputs immediately (this helps avoid undefined behavior)
- As a way of documenting your code

`assert` Demo

code expert Code Example "Debugging with Assert"

Questions?

4. PRE and 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 length){
    return height*length;
}
```

PRE and POST Conditions

Questions: What would be sensible conditions here?

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

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?

PRE and POST Conditions I

Find sensible PRE and POST conditions for this function

```
// 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 and POST Conditions I (Solution)

Possible Solution

```
// 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 and POST Conditions II

Find sensible PRE and POST conditions for this function

```
// 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 and POST Conditions II (Solution)

Possible Solution

```
// 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;
}
```

5. 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;
}
```

What is the output going to be (ignoring possible over- and underflows)?

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;
}
```

What is the output going to be (ignoring possible over- and underflows)? **Solution**: i^7

Bug hunt

```
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;
}
```

Find at least 3 mistakes in this program.

Bug hunt (Solution)

1. `g()` is not yet known to `f()`, since scope of `g()` starts later
2. There's no `%`-operator for `double`
3. `h()` does not "see" the variable `result`, since it is not in its scope
4. `h()` has no return value even though there should be one
5. `h()` is called without an argument

Number of Divisors

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

Example

6 has 4 divisors, namely $1, 2, 3, 6$

Number of Divisors (Solution)

```
// 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;
}
```

Questions?

6. Exam Question

Exam Relevant?

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

Exam Relevant?

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

Questions?

7. Headers and Namespaces

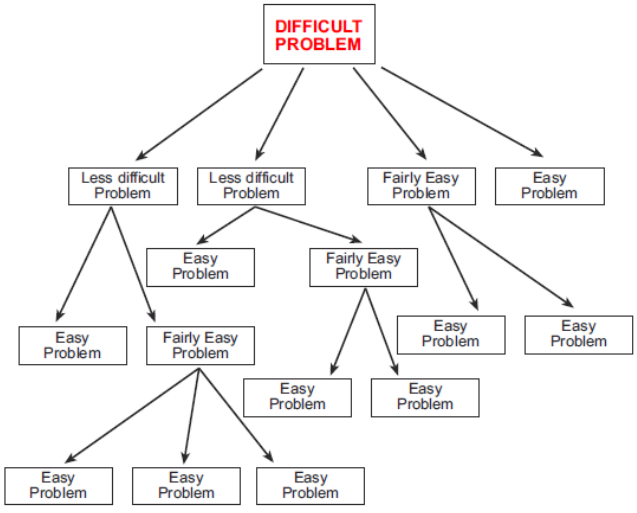
Headers and Namespaces

Live Demo Code Example "CPP Headers & Namespaces"

Questions?

8. Stepwise Refinement

Basic Idea


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

- *Don’t start right away*
- Identify the easier subproblems

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 in $[a, b]$?

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"

```cpp
#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
  unsigned int count = count_perfect_numbers(a, b);

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

  return 0;
}
```

Questions?

9. Old Exam Question

Exam Question "Type and Value"

Provide the type and value of variable `c`[4]

[4] Remark to Type and Value Questions: The keyword **`auto`** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

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

[4] Remark to Type and Value Questions: The keyword **auto** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

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

Solution

[4] Remark to Type and Value Questions: The keyword **auto** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

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

Solution

`int`,

[4] Remark to Type and Value Questions: The keyword **`auto`** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

Solution

`int`, `1`

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

Solution

[4] Remark to Type and Value Questions: The keyword **`auto`** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

Solution

`int`, 1

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

Solution

`double`,

[4] Remark to Type and Value Questions: The keyword **`auto`** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question "Type and Value"

Provide the type and value of variable c[4]

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

Solution

`int`, 1

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

Solution

`double`, 9.2

[4] Remark to Type and Value Questions: The keyword **auto** means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*"

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*"
   True, namely $1.01 \cdot 2^0$

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*" **True**, namely $1.01 \cdot 2^0$
2. "There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$"

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*"
   True, namely $1.01 \cdot 2^0$
2. "There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$"
   True, the smallest number that can be represented is 0.5 (i.e., $1.0 * 2^{-1}$)

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*"
   True, namely $1.01 \cdot 2^0$
2. "There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$"
   True, the smallest number that can be represented is 0.5 (i.e., $1.0 * 2^{-1}$)
3. "3.25 can be represented exactly in the F^*"

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question F^*

Let F^* be the following normalized floating point system[5]

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

True of False?

1. "1.25 can be represented exactly in the floating point system F^*"
   True, namely $1.01 \cdot 2^0$
2. "There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$"
   True, the smallest number that can be represented is 0.5 (i.e., $1.0 * 2^{-1}$)
3. "3.25 can be represented exactly in the F^*"
   False, $3.25 = 1.101 \cdot 2^1$ would require precision $p \geq 4$

[5] Reminder: the precision (number of digits) includes the leading bit.

Exam Question "Loop"

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

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

std::cout << sum;
```

Which statement describes the output best?

Exam Question "Loop"

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

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

std::cout << sum;
```

Which statement describes the output best?

- 17
- 8
- Never terminates
- 18

Exam Question "Loop Termination"

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

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

std::cout << sum;
```

Answer:

Exam Question "Loop Termination"

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

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

std::cout << sum;
```

Answer: It never terminates!

Exam Question "Loop Termination"

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

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

std::cout << sum;
```

Answer: It never terminates!

- Division of two positive ints cannot be negative
  $\Rightarrow$ `sum >= 0` is always true!
- After the first execution of the do block: `i > sum`.
  `sum` is monotonically decreasing, `i` is monotonically increasing.
  $\Rightarrow$ `i > sum` is always true.

10. Outro

General Questions?

See you next time

Have a nice week!