Exercise 6 – Functions

Informatik I für Mathematiker und Physiker (HS 2015)

Yeara Kozlov

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Agenda

- HW #4 Feedback
- Functions
  - Definition and Declaration
  - The Standard Library
  - Exercises
  - Stepwise Refinement
- HW #5 Discussion

Agenda

- **HW #4 Feedback**
- Functions
  - Definition and Declaration
  - The Standard Library
  - Exercises
  - Stepwise Refinement
- HW #5 Discussion

HW #4 Feedback

- Indentation
- Block comments
- Nested for loops
- Integer overflow

HW #4 Feedback - Indentation

```
int main()
{
    if (condition){
        statements;
    }

    for (int i = 0; i < 5; i++)
    {
        if (anotherCondition)
        {
            moreStatements;
        }
    }
}
```

HW #4 Feedback - Block Comments

```
// this is a very long,
// multi line
// comment

if (condition){
    statements;
}

/* This is an easier way to
    comment out a lot of text */
int i = 0;
i++;

/* You can also comment out code
    blocks */
/*
i+=2;
i-=2;
*/
```

HW #4 Feedback - Nested for Loops

```
int main()
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < i; j++)
        {
            std::cout << i * j;
        }
    }
}
```

- Different for loops - different iterators
- Do not modify the iterator of the outer loop within the inner loop
- Usual rules of thumb of for loops

HW #4 Feedback - Integer Overflow

- Very large (usually) negative values
- For example, dec2bin representation
- Can this exercise be solved without storing numbers?

Agenda

- HW #4 Feedback
- **Functions**
  - **Definition and Declaration**
  - The Standard Library
  - Exercises
  - Stepwise Refinement
- HW #5 Discussion

Functions - Structure

return type   function name   argument type   argument

```
int main ( int argc, char* argv[] )
{
        cout << "Hello, World!" << endl;
        return 0;
}
```

argument list

function body

- **function name:** name of the function
- **function body:** statements to be executed
- **argument:** variable whose value is **passed into function body** from the outside
- **argument type**: type of the argument
- **return value:** value that is **passed to the outside** after function call
- **return type:** type of the return value (`void` if there is no return value)

Functions - Advantages

- Readability
- Code reuse and maintenance
- Abstraction

Functions - Definition and Declaration

- Simple functions can be defined once and used everywhere

```
int squared (const int a) {
  return a*a;
}
```

- What happens in more complex cases?

```
void f (...) { //definition of f
    g(...);
}

void g (...) { //definition of g
    f(...);
}
```

Functions - Definition and Declaration

- This code will never compile - we have to use a *forward declaration*

```
void g (…);      //declaration of g

void f (...) { //definition of f
    g(...);
}

void g (...) { //definition of g
    f(...);
}
```

- Declaration have semi-colons ; no curly brackets { }
- Only have the function signature - no functionality

Agenda

- HW #4 Feedback
- **Functions**
  - Definition and Declaration
  - **The Standard Library**
  - Exercises
  - Stepwise Refinement
- HW #5 Discussion

The Standard Library

- *"collection of classes and functions, which are written in the core language and part of the C++ ISO Standard itself"*

- you can access it through **includes** like iostream, climits, cmath, …

- features of the C++ Standard Library are declared within the std namespace
  - call every function using the std:: specifier

The Standard Library - Motivation

- Written and published once, the code can be used by anyone
- (Usually) Programmed by an expert in the field.
- Easily maintainable
- Less time wasted maintaining code implies more time to write especially fast and efficient code

The Standard Library - cmath

- `cmath` provides numerical approximations of abstract mathematical numbers such as sqrt, powers, sin, cos
- `cmath` implementation uses floating point numbers
  - some numbers cannot be represented

The Standard Library - cmath

```
#include <iostream>
#include <cmath>

int main () {

  std::cout << std::pow(3.3, 6.5) << "\n"  // computes 3.3 ^ 6.5

            << std::pow(3, 6)     << "\n"  // computes 3.0 ^ 6.0, all arguments are implicitly
                                           // converted to double

            << std::sqrt(9.1)     << "\n"  // computes the square root of 9.1, the argument has
                                           // to be >=0

            << std::abs(-3.0)     << "\n"; // computes 3.0, the absolute value of the argument

  return 0;
}
```

The Standard Library - cmath

- `std::sqrt` returns the numerical approximation of an abstract mathematically correct value.
- How can we measure its correctness?

$$|\mathrm{std::sqrt}(x) * \mathrm{std::sqrt}(x) - x| = \mathrm{err}$$

- Exercise: read a `double` value from the user and computes the error measure for this value.

The Standard Library - cmath

```cpp
#include <iostream>
#include <cmath>
#include <cassert>

int main () {

  double x;
  std::cin >> x;
  assert(x >= 0);

  const double sqrtx = std::sqrt(x);
  std::cout << std::abs(sqrtx*sqrtx - x) << "\n";

  return 0;
}
```

The Standard Library - algorithm

```cpp
#include <iostream>
#include <algorithm>

int main () {

  std::cout << std::min (3.5, 5.1) << "\n"   // outputs 3.5
            << std::max (4.3, 7.0) << "\n";  // outputs 7.0
  return 0;
}
```

Agenda

- HW #4 Feedback
- **Functions**
  - Definition and Declaration
  - The Standard Library
  - **Exercises**
  - Stepwise Refinement
- HW #5 Discussion

Exercise 3

- Fix the **problems** in the following functions.
- Then add suitable **PRE- and POST-conditions.**

1. Function:

```
bool is_even (const int i)
{
    if (i % 2 == 0) return true;
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: just a return value for even inputs

1. Function:

```
bool is_even (const int i)
{
    if (i % 2 == 0) return true;
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: just a return value for even inputs
- Fix: e.g. direct return of `i % 2 == 0`

1. Function:

```
bool is_even (const int i)
{
    if (i % 2 == 0) return true;
}
```

```
bool is_even (const int i)
{
    return (i % 2 == 0);
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: just a return value for even inputs
- Fix: e.g. direct return of `i % 2 == 0`

1. Function:

```
bool is_even (const int i)
{
    if (i % 2 == 0) return true;
}
```

```
bool is_even (const int i)
{
    return (i % 2 == 0);
}
```

PRE-Condition: (not needed)

POST-Condition:

```
// POST: return value is true if and only
//       if i is even
```


(From: Script Exercise 80)

Exercise 3

- Fix the **problems** in the following functions.
- Then add suitable **PRE- and POST-conditions.**

2. Function:

```
double inverse (const double x) {
   double result;
   if (x != 0.0)
        result = 1.0 / x;
   return result;
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: no return value for $x = 0$

2. Function:

```
double inverse (const double x) {
    double result;
    if (x != 0.0)
         result = 1.0 / x;
    return result;
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: no return value for `x=0`
- Fix: `x != 0.0` as PRE-condition (and `assert`)

2. Function:

```
double inverse (const double x) {
    double result;
    if (x != 0.0)
        result = 1.0 / x;
    return result;
}
```

```
// PRE: x != 0.0
// POST: ...
double inverse (const double x) {
    assert(x != 0.0);
    return 1.0 / x;
}
```


(From: Script Exercise 80)

Exercise 3

- Problem: no return value for `x=0`
- Fix: `x != 0.0` as PRE-condition (and `assert`)

2. Function:

```
double inverse (const double x) {
    double result;
    if (x != 0.0)
        result = 1.0 / x;
    return result;
}
```

```
// PRE: x != 0.0
// POST: ...
double inverse (const double x) {
    assert(x != 0.0);
    return 1.0 / x;
}
```

PRE-Condition: `// PRE: x != 0.0`

POST-Condition: `// POST: return value is 1/x`

(From: Script Exercise 80)

Exercise 3

Another solution:

`else` with special return value

```
double inverse (const double x)
{
    double result;
    if (x != 0.0)
           result = 1.0 / x;
    else
           result = 0.0;
    return result;
}
```


(From: Script Exercise 80)

Exercise 3

Another solution:

`else` with special return value

```
double inverse (const double x)
{
    double result;
    if (x != 0.0)
        result = 1.0 / x;
    else
        result = 0.0;
    return result;
}
```

PRE-Condition: (not needed)

POST-Condition:

```
// POST: return value is 1/x if x!=0.0
//       return value is 0.0 else
```


(From: Script Exercise 80)

Exercise 4

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

```
f(i)
```

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

```
i*i
```

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

```
i*i
```

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

`f(f(i))`

`f(i)`

```cpp
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

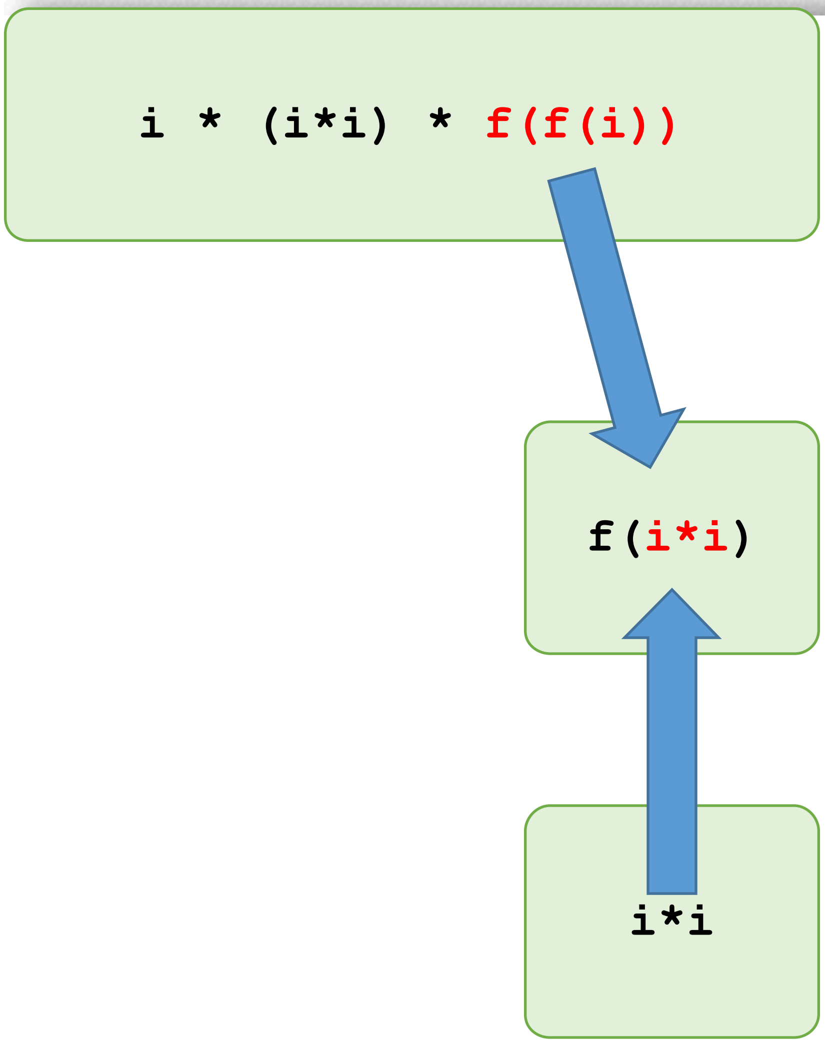
`f(f(i))`

`i*i`

```
#include <iostream>

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

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

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

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

(From: Script Exercise 81)

Exercise 4




```
#include <iostream>

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

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

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

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

(From: Script Exercise 81)

Exercise 4

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

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

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

```cpp
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

```
#include <iostream>

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

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

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

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


(From: Script Exercise 81)

Exercise 4

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

This is
i^7

```
#include <iostream>

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

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

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

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

Agenda

- HW #4 Feedback
- **Functions**
  - Definition and Declaration
  - The Standard Library
  - Exercises
  - **Stepwise Refinement**
- HW #5 Discussion

Stepwise Refinement

- Approaching for writing (more complex) programs:
  - Where do we start?
  - What functions will we need?
  - How to structure the program?
  - …
- Stepwise Refinement → **stepwise approach**

Stepwise Refinement - Idea

- From coarse-grained to fine-grained.

Procedure

1. **Outline**
   coarse-grained structure using comments

2. **Refine**
   comments repeatedly with
   - finer-grained comments
   - code
   - (hypothetical) function calls

Example - Stepwise Refinement

Goal:

Write a simple program which computes the mid-point between two points on a line.

Example - Stepwise Refinement

Coarse-grained structure

```
int main () {
    // input the two points

    // compute mid-point

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // input the two points

    // compute mid-point

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // compute mid-point

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // compute mid-point

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation of mid-point
    const double m = mid_point(a, b);

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation of mid-point
    const double m = mid_point(a, b);

    // output mid-point
}
```

Hypothetical function

(implement later)

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation of mid-point
    const double m = mid_point(a, b);

    // output mid-point
}
```

Example - Stepwise Refinement

Refine

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation of mid-point
    const double m = mid_point(a, b);

    // Output of computed result
    std::cout << "The mid-point is: " << m << "\n";
}
```

Example - Stepwise Refinement

Cleanup

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation and output
    std::cout << "The mid-point is: " << mid_point(a, b)
              << "\n";
}
```

Example - Stepwise Refinement

Cleanup

```
int main () {
    // Input of values
    std::cout << "Input the first point: ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation and output
    std::cout << "The mid-point is: " << mid_point(a, b)
              << "\n";
}
```

Now:

Implement hypothetical functions

Example - Stepwise Refinement

Cleanup

```
int main () {
    // Input of valu
    std::cout << "Inp                              ";
    double a;
    std::cin >> a;

    std::cout << "Input the second point: ";
    double b;
    std::cin >> b;

    // Computation and output
    std::cout << "The mid-point is: " << mid_point(a, b)
              << "\n";
}
```

Using Stepwise Refinement

Now:

Implement hypothetical functions

Example - Midpoint Computation

Coarse-grained structure

```
// POST: returns the mid-point between a and b.
double mid_point (double a, double b) {

    // compute mid-point

    // return
}
```

Example - Midpoint Computation

Refine

```
// POST: returns the mid-point between a and b.
double mid_point (double a, double b) {

    // compute mid-point

    // return
}
```

Example - Midpoint Computation

Refine

```
// POST: returns the mid-point between a and b.
double mid_point (double a, double b) {

    const double mid_point = (a + b) / 2;

    // return
}
```

Example - Midpoint Computation

Refine

```
// POST: returns the mid-point between a and b.
double mid_point (double a, double b) {

    const double mid_point = (a + b) / 2;

    // return
}
```

Example - Midpoint Computation

Refine

```
// POST: returns the mid-point between a and b.
double mid_point (double a, double b) {

    const double mid_point = (a + b) / 2;

    return mid_point;
}
```

Example - Midpoint Computation

Cleanup

```
// POST: returns the mid-point between a and b.
double mid_point (const double a, const double b) {
    return (a + b) / 2;
}
```

Stepwise Refinement - Exercises

Given: two natural input numbers: lower <= upper

Print all palindromes in the set {lower, lower+1, ..., upper-1, upper}.

A palindrome is a number that is equal to the number with digits reversed.

Examples: 121, 9537359

Agenda

- HW #4 Feedback
- Functions
  - Definition and Declaration
  - The Standard Library
  - Exercises
  - Stepwise Refinement
- **HW #5 Discussion**

References*

- Used as an alias for another variable
- Must be **initialized** with a **lvalue**
- Can only be initialized once

```
int a = 3;
int& b = a;
std::cout << b << "\n"; // Output: 3
a = 4;
std::cout << b << "\n"; // Output: 4
b = 2;
std::cout << a << "\n"; // Output: 2
```