Assert

`assert`	**sofortiges Stoppen** des Programms bei Verletzung einer Bedingung (zu Testzwecken)

Erfordert: `#include<cassert>`

Wenn das fertige Programm veröffentlicht werden soll, kann man die `assert`-Befehle bequem deaktivieren.

```
int a;
int b;
std::cin >> a >> b; // read two int values from user
assert(b != 0); // prevent division by 0
std::cout << a / b << "\n";
```

Figure: Aus: Summary 6, Programmier-Befehle - Woche 6

Assert

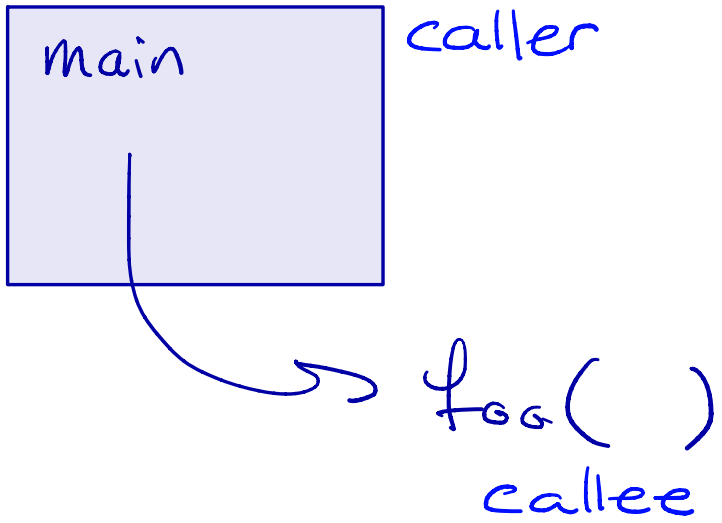
Where are asserts useful?

- In long programs to have a better overview.
- To catch wrong (user-)inputs immediately and avoid undefined behaviour later on.
- As an additional form of documentation when multiple programmers are working on the same code.

To give more information on what the error is:

```
assert(condition && "error message");
```

Recap: Functions



return type
name
args

```
bool foo(int a, int b){
    // body
    return true;
}

void bar(bool a, char b, double c){
    if(a){
        return;
    }
    // statements
}
```

For a function of return type void , a return statement is not strictly necessary. If the end of such a function is reached without encountering a return statement, control is passed to the caller as if a return statement without an expression were encountered.

Stepwise Refinement

```
bool move_left(...) {
  //
  return false;
}
```

Writing code:

- Break down large problems into smaller steps that can solve the problem.
- Turn these smaller steps into code blocks, e.g. functions.
- Once you have a good framework, you can start filling the code for the individual functions.
- Check if individual functions work correctly, then check if the whole code works.

Debugging:

- Find out step-by-step in which function, in which region etc does the code stop working the way we expect?

Documentation: Pre- and Post-conditions

From the C++ Google Style Guide:

"Almost every function declaration should have comments immediately preceding it that describe what the function does and how to use it. "

Documentation: Pre- and Post-conditions

Wie soll eine PRE/Post condition sein?

- PRE: So weit wie möglich, Ihr wollt den Nutzen der Funktion nicht einschränken.
- POST: Möglichst eng, Ihr wollt genau wissen was passiert/was die Funktion zurück gibt.

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:

PRE-Condition:
(not needed)

POST-Condition:

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

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.

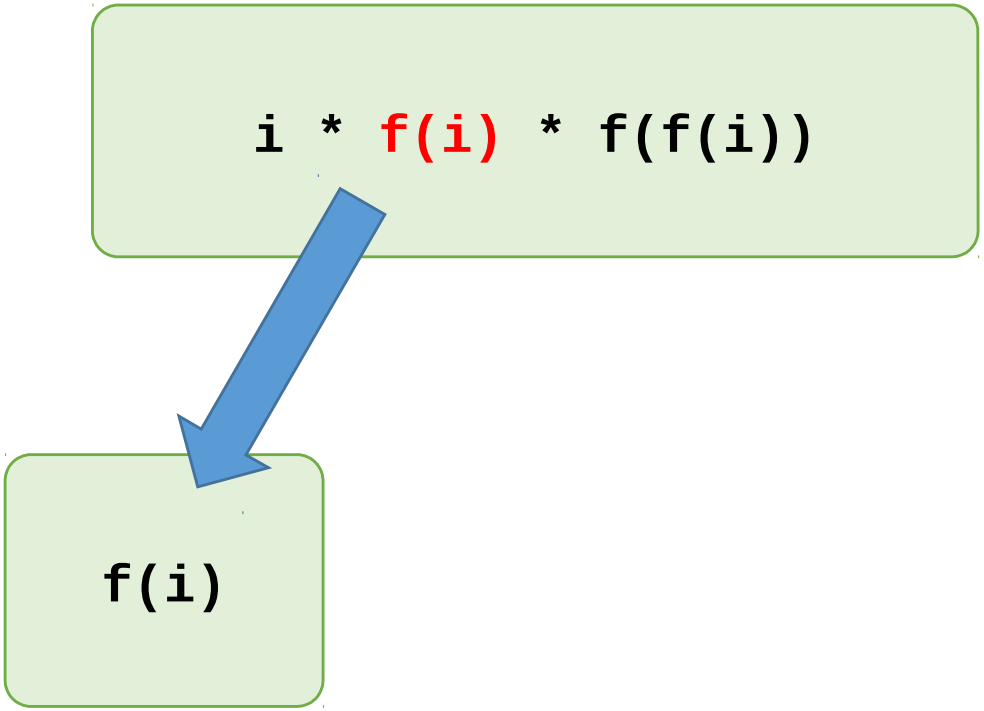
```
#include <iostream>

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))
```

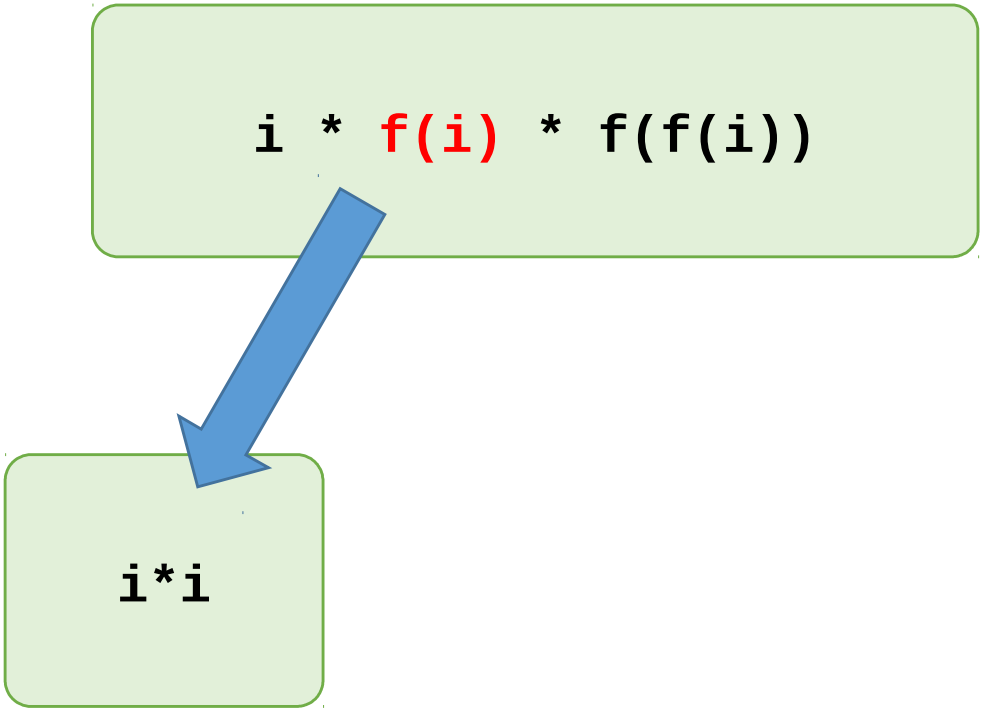
```
#include <iostream>

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



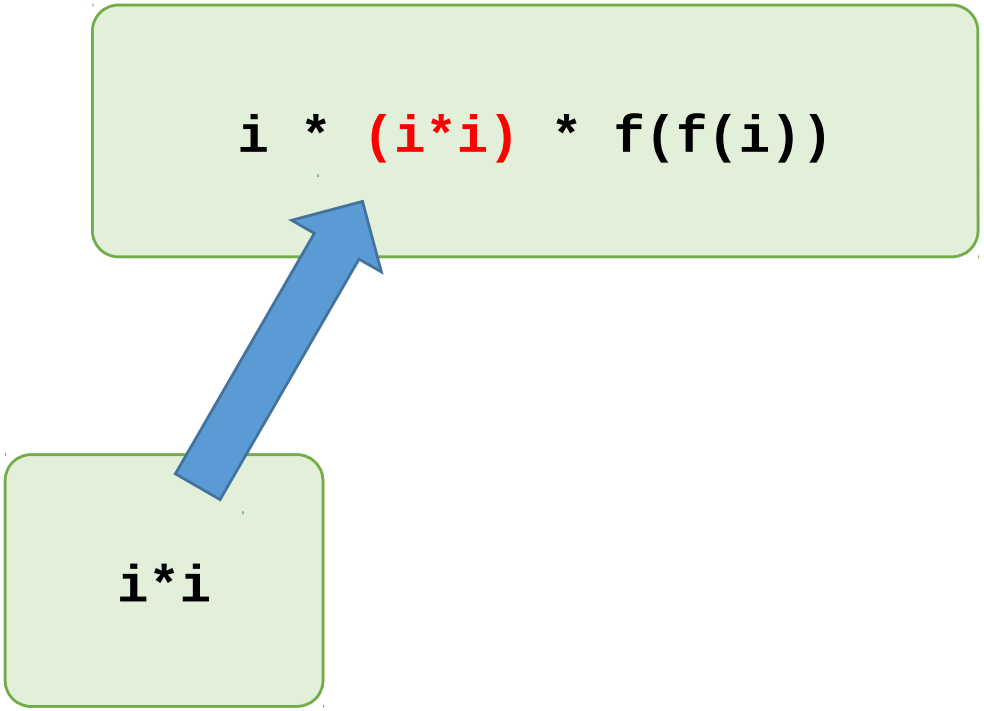

```
#include <iostream>

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



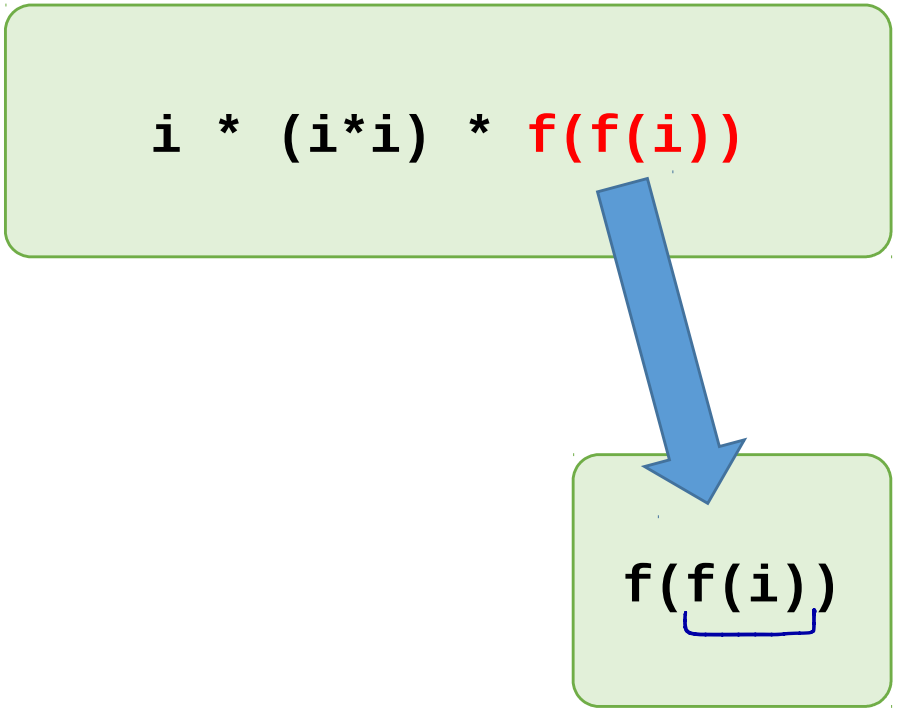

```
#include <iostream>

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



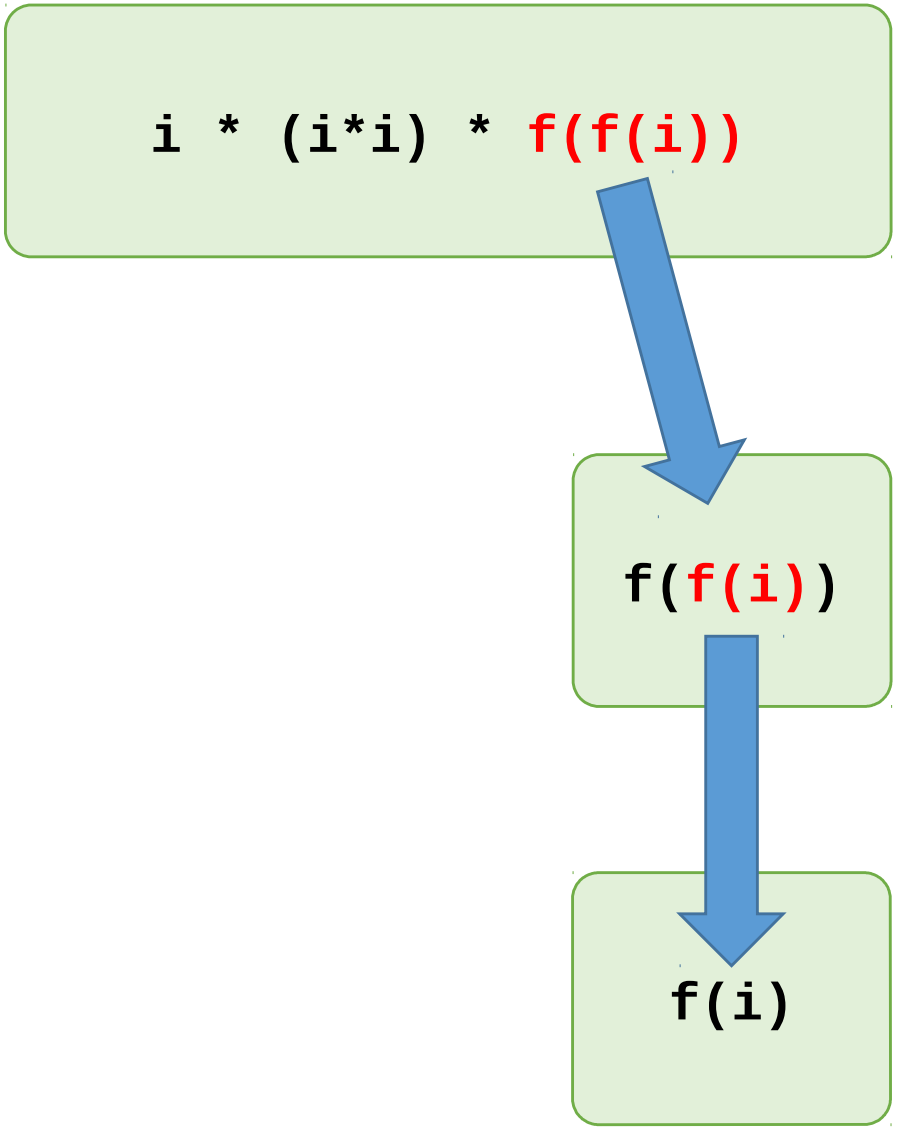

```
#include <iostream>

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



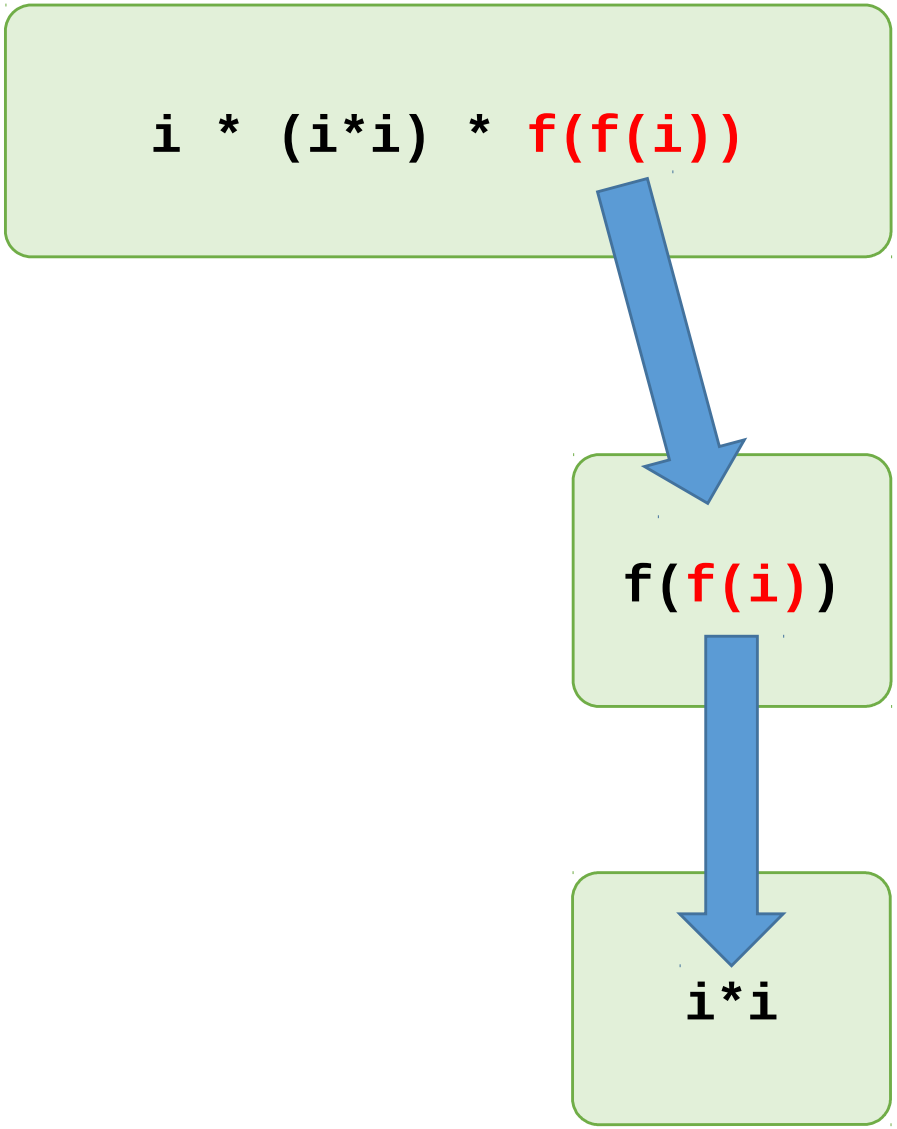

```
#include <iostream>

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



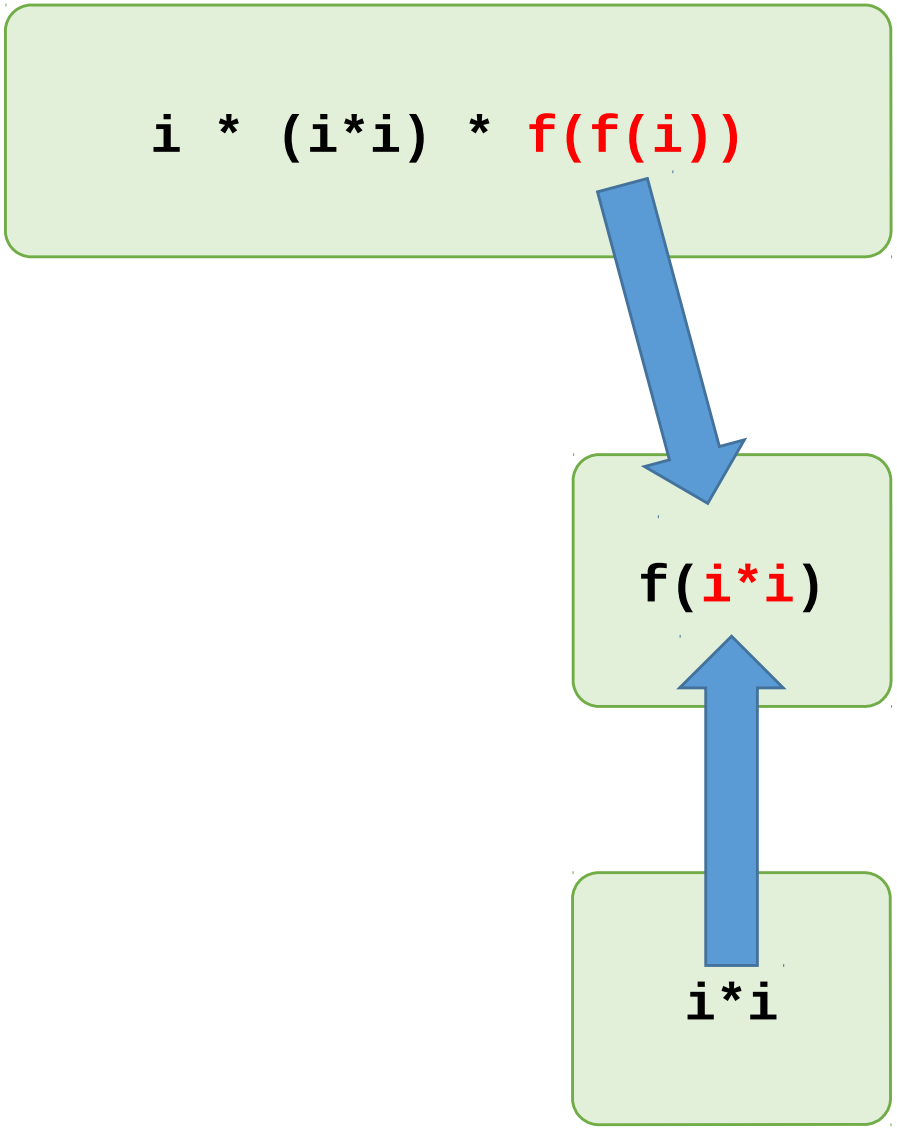

```
#include <iostream>

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



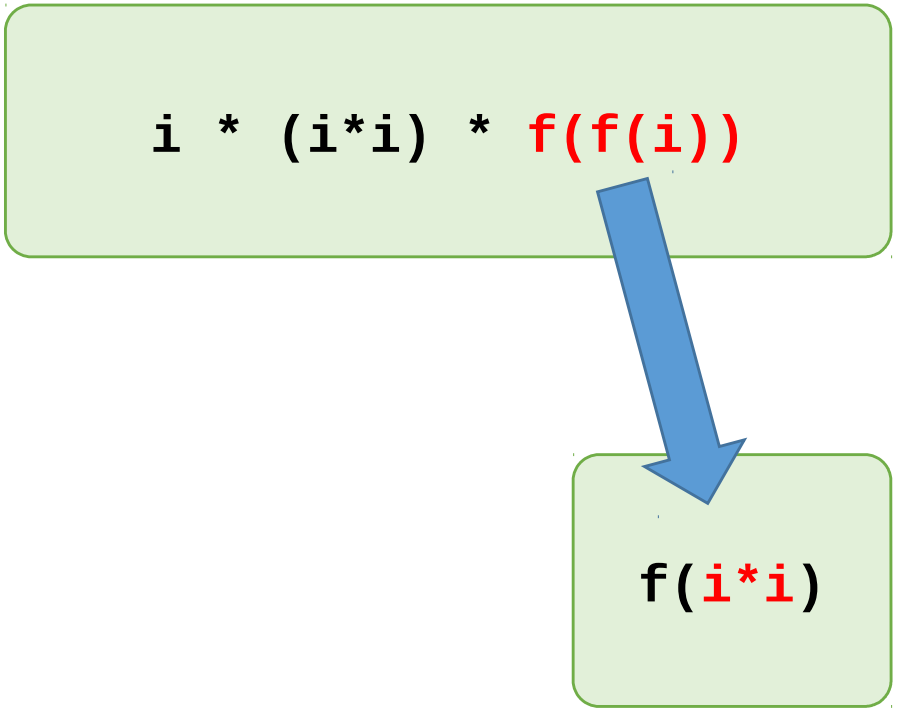

```
#include <iostream>

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



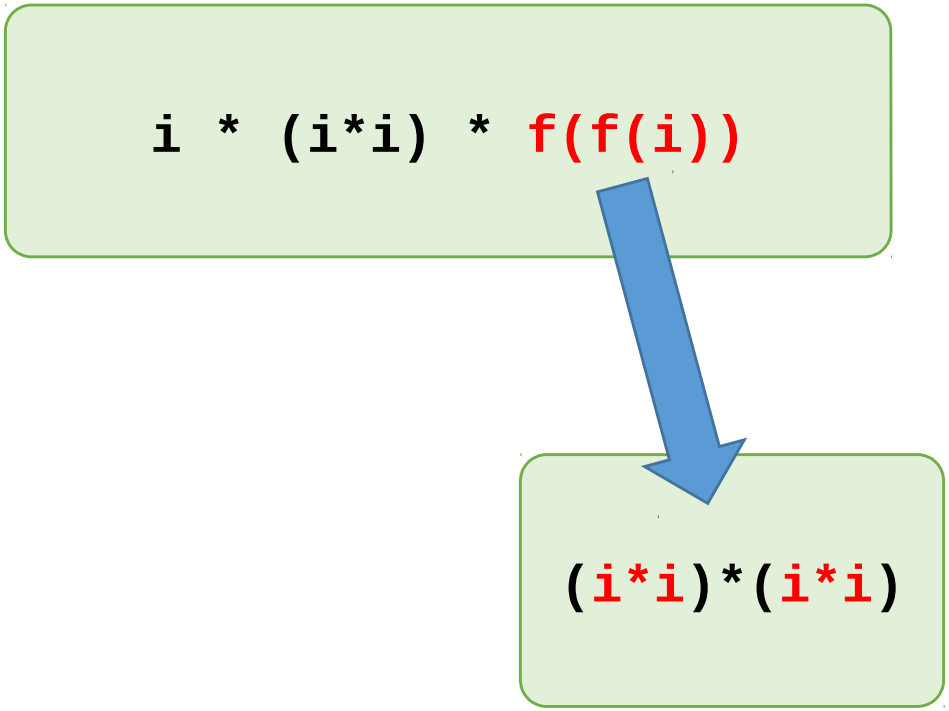

```
#include <iostream>

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



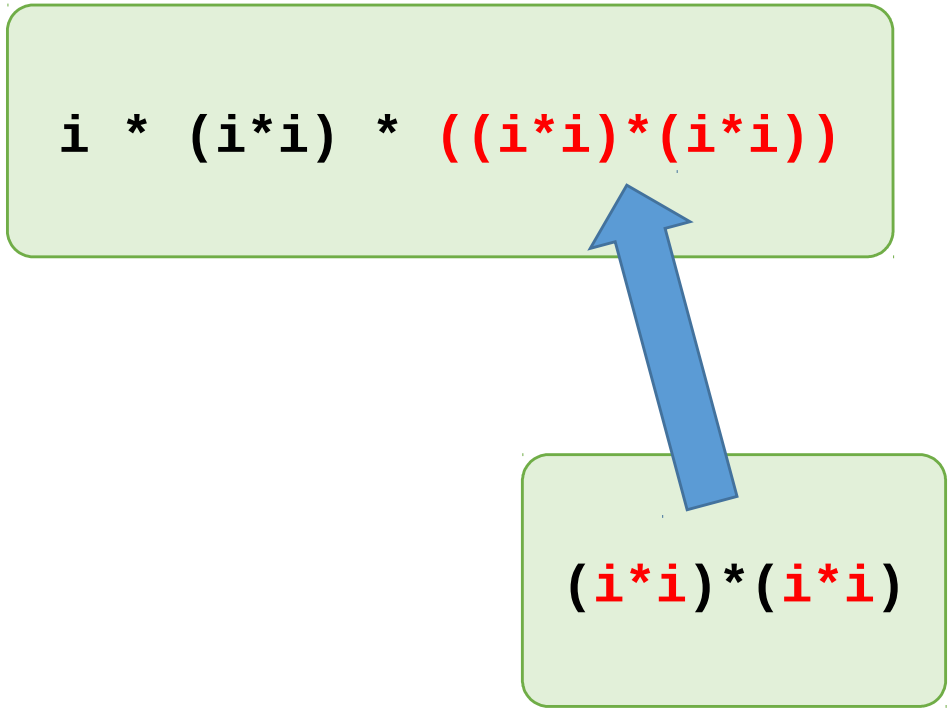

```
#include <iostream>

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;
}
```

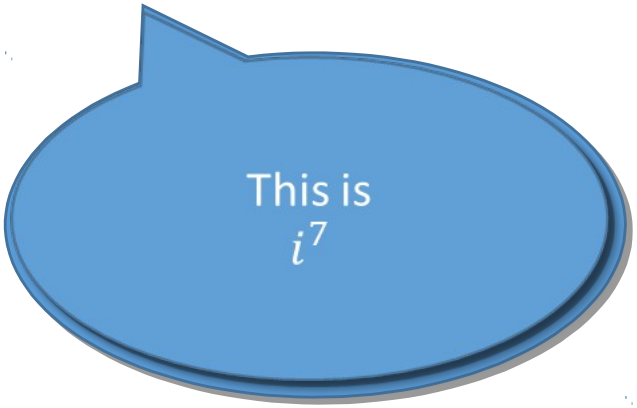
Exercise 2

```
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 k( int, int );
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;
}
```

k(int a, int b)

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;
}
```

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

```
# 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`

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

`result` is out-of-scope

(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 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;
}
```

Informatik I - Exercise Session
Past Exam Questions

[Exam 2022-08 MAVT] Expression Evaluation

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.

1. Provide type and value of variable c.

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

int

9 x 5

(45 + 1)

46 %5

= 1

2. Provide type and value of variable c.

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

double

46 / 5

9.2

[Exam 2022-08 MAVT] Expression Evaluation

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.

1. Provide type and value of variable c.

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

`int`

2. Provide type and value of variable c.

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

[Exam 2022-08 MAVT] Expression Evaluation

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.

1. Provide type and value of variable c.

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

`int , 1`

2. Provide type and value of variable c.

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

[Exam 2022-08 MAVT] Expression Evaluation

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.

1. Provide type and value of variable c.

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

`int , 1`

2. Provide type and value of variable c.

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

`double`

[Exam 2022-08 MAVT] Expression Evaluation

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.

1. Provide type and value of variable c.

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

`int , 1`

2. Provide type and value of variable c.

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

`double , 9.2`

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

[1.xx]xx

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.

  True $1.01 \cdot 2^0$

- There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$.

  True $1.00 \cdot 2^{-1}$

- 3.25 can be represented exactly in the floating point system $F*$.

  False

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.
  TRUE
- There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$.
- 3.25 can be represented exactly in the floating point system $F*$.

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

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

$0.0625 = \frac{1}{16}$
$0.25 = \frac{1}{4}$

- 3.25 can be represented exactly in the floating point system $F*$.

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.
  TRUE , $1.01 * 2^0$
- There is no number $Z \in F^*$ such that $0.0625 < Z < 0.25$.
  TRUE
- 3.25 can be represented exactly in the floating point system $F*$.

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.
  TRUE , $1.01 * 2^0$
- 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., $1.0 * 2^{-1}$)
- 3.25 can be represented exactly in the floating point system $F*$.

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.
  TRUE , $1.01 * 2^0$
- 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., $1.0 * 2^{-1}$)
- 3.25 can be represented exactly in the floating point system $F*$.
  FALSE

[Exam 2022-08 MAVT] Normalized Floating Point Systems

Answer the following questions regarding the normalized floating point system F*.

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

Reminder: For F*, the precision (number of digits) includes the leading bit.

True or false?

- 1.25 can be represented exactly in the floating point system $F*$.
  TRUE , $1.01 * 2^0$
- 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., $1.0 * 2^{-1}$)
- 3.25 can be represented exactly in the floating point system $F*$.
  FALSE , $3.25 = 1.101 * 2^1$ would require precision $p \geq 4$

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```
1 int sum = 17;
2 int i = 1;
3
4 do {
5   i += sum;
6   sum = sum / 2;
7 } while (i > sum && sum >= 0);
8
9 std::cout << sum;
```

→ i = 18
sum 8

→ i = 26
sum 4

Which statement describes the output best?

- ○ 17
- ○ 8
- ✗ Never terminates
- ○ 18

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```
1 int sum = 17;
2 int i = 1;
3
4 do {
5   i += sum;
6   sum = sum / 2;
7 } while (i > sum && sum >= 0);
8
9 std::cout << sum;
```

Which statement describes the output best?

- ○ 17
- ○ 8
- ○ Never terminates CORRECT
- ○ 18

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```
1 int sum = 17;
2 int i = 1;
3
4 do {
5   i += sum;
6   sum = sum / 2;
7 } while (i > sum && sum >= 0);
8
9 std::cout << sum;
```

Which statement describes the output best?

- ○ 17
- ○ 8
- ○ Never terminates CORRECT
- ○ 18

- Division of two positive `ints` cannot be negative.
  ⇒ `sum >= 0` is always `true`

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```cpp
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 CORRECT
- ○ 18

- Division of two positive `int`s cannot be negative.
  $\Rightarrow$ `sum >= 0` is always `true`
- After the first execution of the `do` block: `i > sum`.

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```cpp
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 CORRECT
- ○ 18

- Division of two positive `int`s cannot be negative. ⇒ `sum >= 0` is always `true`
- After the first execution of the `do` block: `i > sum`. `sum` is monotonically decreasing, `i` is monotonically increasing.

[Exam 2022-08 252-08(47/48/56)] Loop Termination

```cpp
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 CORRECT
- ○ 18

- Division of two positive `int`s cannot be negative.
  ⇒ `sum >= 0` is always `true`
- After the first execution of the `do` block: `i > sum`.
  `sum` is monotonically decreasing, `i` is monotonically increasing.
  ⇒ `i > sum` is always `true`

C and Cpp header files: https://www.geeksforgeeks.org/header-files-in-c-cpp-and-its-uses/

Hier noch ein StackOverflow thread dazu: https://stackoverflow.com/questions/1305947/why-does-c-need-a-separate-header-file

=> Schlussendlich eine Mischung aus
- -> historisch gewachsen wegen C
- -> Möglichkeit gewisse Sachen schneller zu machen beim compilen

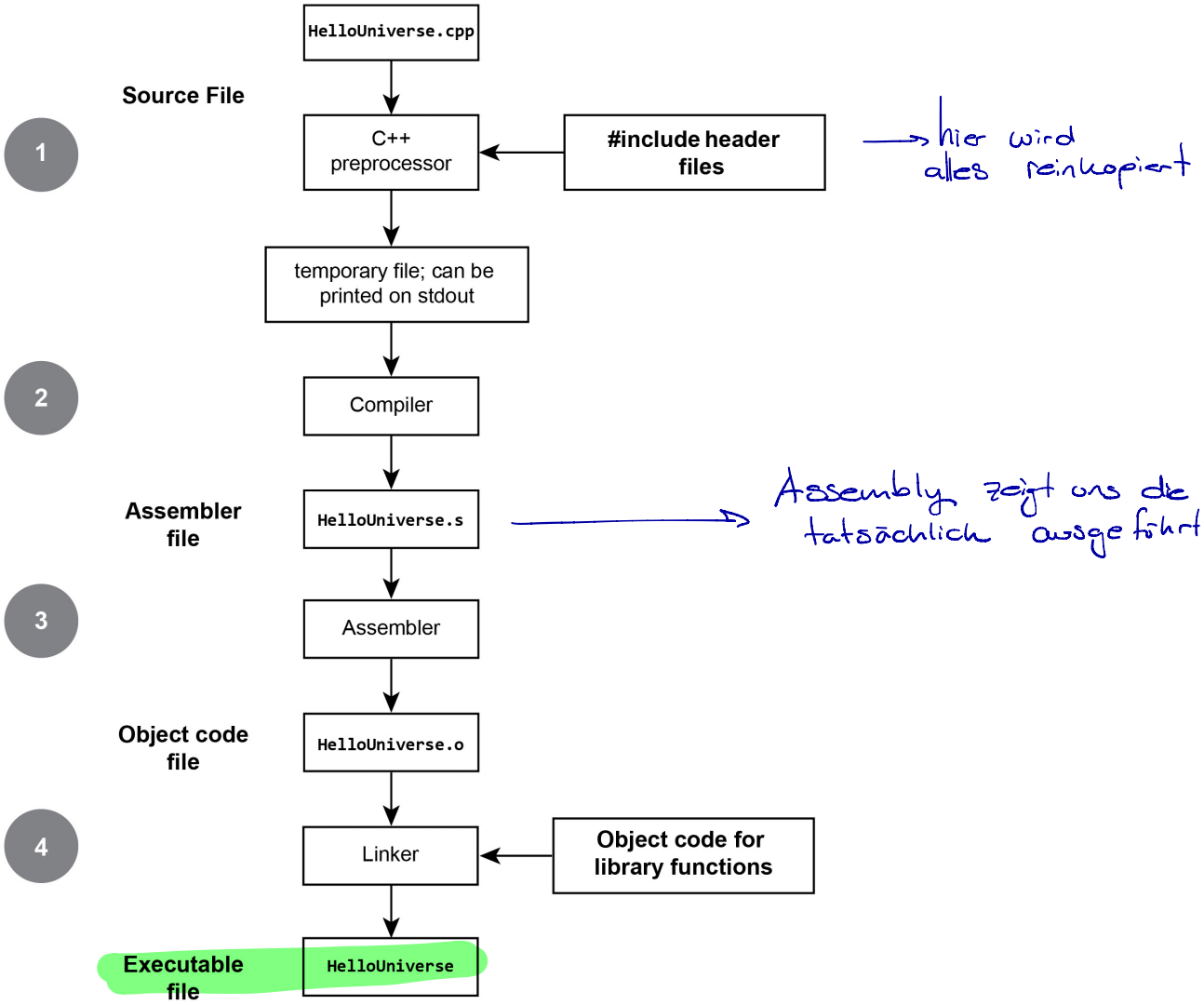
=> Ich denke sehr viel ist historisch gewachsen (1972 für C, 1985 für C++) hier nach meinem Wissen andere "modernere" wege heutige compiled languages wählten

=> Nutzen für Compiler: Der gleiche Code muss nicht bei jedem #include jedes Mal compiliert werden.

=> Nutzen für uns: Hauptsächlich die Struktur/Ordnung des Codes

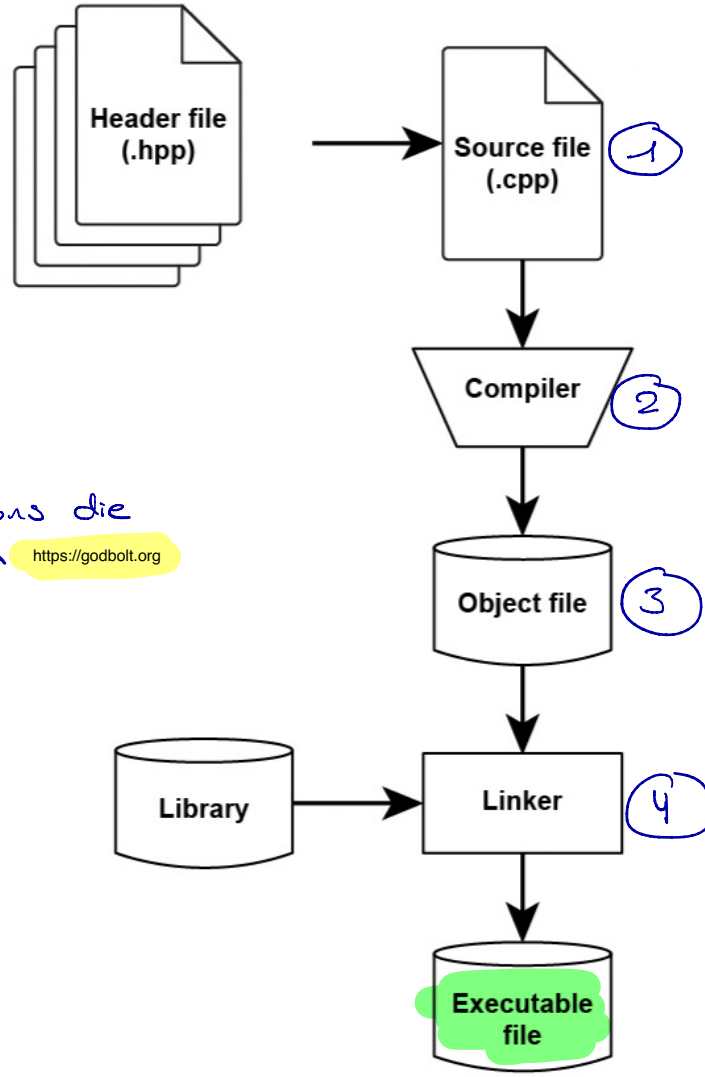
Build Pipeline / Build Process: C++ Compilation Model

https://subscription.packtpub.com/book/programming/9781789801491/1/ch01lvl1sec03/the-c-compilation-model



→ hier wird alles reinkopiert

Assembly zeigt uns die instructions die tatsächlich ausgeführt werden https://godbolt.org



↳ "runnable" i.e. dieser File wird ausgeführt wenn ihr in CodeExpert "play" drückt.

↳ (können z.B. die Endung ".exe" haben aber müssen auch gar keine Endung haben)