Informatik I - Exercise Session

Hidden Test Cases, Assert, Functions, Headers and Namespaces, Stepwise Refinement, Past Exam Questions

[code]expert Changes

Hidden test cases:

- Success as usual
- Failure: in- and output are hidden for you
- Edge cases, special values, . . .

[code]expert Changes

Hidden test cases:

- Success as usual
- Failure: in- and output are hidden for you
- Edge cases, special values, . . .

Persistent input:

1. Save inputs in file `input.txt`
2. Run program
3. Type `f` (without spaces, stands for “file”) and then enter
4. The contents of the file `input.txt` are read and the corresponding outputs displayed

The persistent input is only for you and won’t impact your score.

Debugging with Assert

Table with explanation for exit codes:

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

Debugging with Assert

Table with explanation for exit codes:
`https://lec.inf.ethz.ch/ifmp/2024/guides/debugging/exit_codes.html`

We have done some debugging using print statements; there is a more elegant way using assert:

```
assert(true);  // does absolutely nothing and continues
assert(false); // 'crashes', i.e. exits immediately with code -6
```

Debugging with Assert

Table with explanation for exit codes:
`https://lec.inf.ethz.ch/ifmp/2024/guides/debugging/exit_codes.html`

We have done some debugging using print statements; there is a more elegant way using assert:

```
assert(true);  // does absolutely nothing and continues
assert(false); // 'crashes', i.e. exits immediately with code -6
```

`asserts` can be useful

- to have better overview in long programs,
- to catch wrong (user) inputs that could lead to erroneous/undefined behaviour,
- to document (for multi-person projects),
- or to enforce pre- and post-conditions.

Functions

Program tracing tutorial:
`https://lec.inf.ethz.ch/ifmp/2024/guides/tracing/calls.html`

Headers and Namespaces

Header files (suffix `.h`) contain namespace, class and functions *declarations*, while the `.cpp` files contain the implementation. Import them using `#include "myHeader.h"`.

Headers and Namespaces

Header files (suffix `.h`) contain namespace, class and functions *declarations*, while the `.cpp` files contain the implementation. Import them using `#include "myHeader.h"`.

This has the simple advantage that you can use everything declared in your header everywhere in your `.cpp` file while not worrying about the declaration order messing up your scopes.

Headers and Namespaces

Header files (suffix `.h`) contain namespace, class and functions *declarations*, while the `.cpp` files contain the implementation. Import them using `#include "myHeader.h"`.

This has the simple advantage that you can use everything declared in your header everywhere in your `.cpp` file while not worrying about the declaration order messing up your scopes.

Namespaces are exactly what you would expect: separated spaces for your (function or class) names. The are used primarily to group declarations and avoid collisions when using the same name multiple times.

Headers and Namespaces

Header files (suffix `.h`) contain namespace, class and functions *declarations*, while the `.cpp` files contain the implementation. Import them using `#include "myHeader.h"`.

This has the simple advantage that you can use everything declared in your header everywhere in your `.cpp` file while not worrying about the declaration order messing up your scopes.

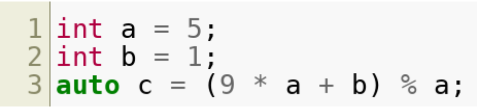
Namespaces are exactly what you would expect: separated spaces for your (function or class) names. The are used primarily to group declarations and avoid collisions when using the same name multiple times.

Don't add `using namespace std;` to your files to avoid these collisions; the namespace `std` is defined by the C++ standard library and contains many things you don't want to mess with. Additionally, when writing `namespace::function()`, it is clear for everyone instantaneously what function from what namespace the code calls.

[Exam 2022-08 MAVT] Expression Evaluation

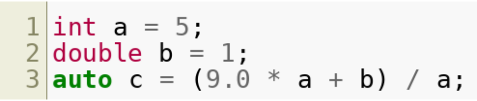
Remark to Type and Value Questions: The keyword `auto` means that the type of the expression is determined by the compiler. In the following it thus stands for the expression type that you need to identify.

1. Provide type and value of variable c.



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

2. Provide type and value of variable c.



```
int a = 5;
double b = 1;
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.

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

`int`

2. Provide type and value of variable c.

```cpp
int a = 5;
double b = 1;
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.

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

int , 1

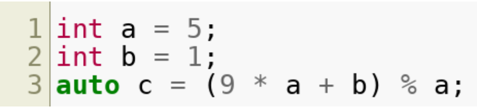
2. Provide type and value of variable c.

```
int a = 5;
double b = 1;
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.

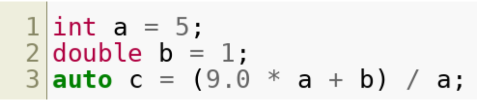
1. Provide type and value of variable c.



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

`int , 1`

2. Provide type and value of variable c.



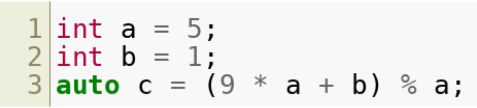
```
int a = 5;
double b = 1;
auto c = (9.0 * a + b) / a;
```

`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.

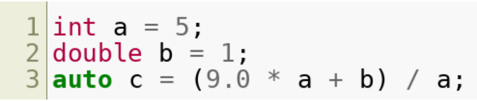
1. Provide type and value of variable `c`.



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

`int , 1`

2. Provide type and value of variable `c`.



```
int a = 5;
double b = 1;
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*.

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

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 $F*$.
- 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
- 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 $0.0625 < Z < 0.25$.
- 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*.

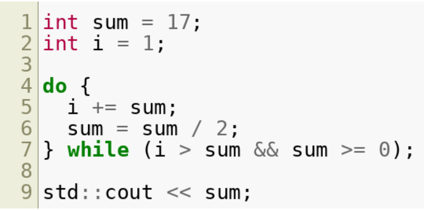
$$F^*(\beta = 2, p = 3, e_{\text{min}} = -1, e_{\text{max}} = 4)$$

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



```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?

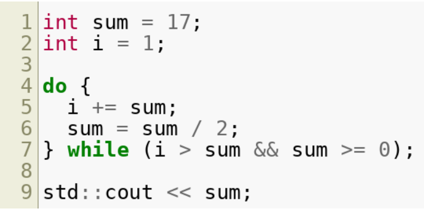
○ 17

○ 8

○ Never terminates

○ 18

[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

[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

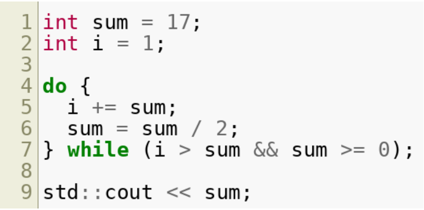
○ 8

○ Never terminates CORRECT

○ 18

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

[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

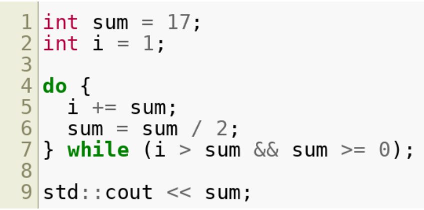
○ 8

○ 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`.

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



```
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`