Exercise 3 – Logical Operators, Branching, Loops

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 #1 feedback
- Expressions
- if statements
- for loop
- const
- assert
- HW#3 hints

Agenda

- **HW #1 feedback**
- Expressions
- if statements
- for loop
- const
- assert
- HW#3 hints

HW #1 Feedback

- Five students with no submissions / users
- Submit each exercise to its specific link in the exercise sheet
- Errors:
  - Compiler errors
  - Output errors
  - Programs that did not perform the required functionality

Agenda

- HW #1 feedback
- **Expressions**
- if statements
- for loop
- const
- assert
- HW#3 hints

Expressions - operators

- Used to compare two values.
- The whole term is called an **expression** (german **Ausdruck**).
- The result of the comparison is either TRUE (1) or FALSE (0).
- Relational operators are evaluated **after** other arithmetic operations.

```
int a=3, b=5;
bool res;
res = a < b; //less
res = a <= b; //less or equal
res = a > b; //greater
res = a >= b; //greater or equal
res = a == b; //equal
res = a != b; //not equal
```

Expressions - = and ==

- „=“ is the **assignment operator**. It **changes the value** of the variable on the left to the value on the right. The result (meaning the value of the expression) is equal to the assignment value (which is **TRUE** for all values not zero)!
- „==“ is the **equality operator**. It evaluates whether the 2 values on the left and the right are **equal**. The result can be **either TRUE or FALSE.**

```
int a=3, b=5;
bool res;
res = (a == b); //“res“ is FALSE, because 3!=5
res = (a = b); //“res“ is TRUE, even though 3!=5.
                //Also, a is set to 5!
```

Logical Operators

- Works on Boolean values (**`bool`** type)

```
bool a = true;
bool b = false;
```

logical **AND**	`a && b`	`== false`
	`a && a`	`== true`
logical **OR**	`a || b`	`== true`
	`b || b`	`== false`
logical **NOT**	`!a`	`== false`
	`!b`	`== true`

Truth Tables

AND	0	1
0	0	0
1	0	**1**

OR	0	1
0	0	**1**
1	**1**	**1**

NOT	
0	**1**
1	**0**

- **x AND y** is TRUE if and only if both x and y are TRUE.
- **x OR y** is FALSE if and only if both x and y are FALSE.
  (**x OR y** is TRUE if either x, y or both are TRUE.)
- **NOT x** is TRUE if x is FALSE, and vice versa.

Expressions - Short Circuit

- Short circuit evaluation guarantees evaluation of the operators && and || from the left to the right.
- The compiler first evaluates the left expression first!
- Easier to evaluate expression should be on the left.

Expressions - Short Circuit

- 2 > 3 && 17u - 55 <= ++x % y

Expressions - Short Circuit

- 2 > 3 && 17u - 55 <= ++x % y
- (2 > 3) && (17u - 55 <= ++x % y)

Expressions - Short Circuit

- 2 > 3 && 17u - 55 <= ++x % y
- (2 > 3) && (17u - 55 <= ++x % y)
- false && (17u - 55 <= ++x % y)

Expressions - Short Circuit

- 2 > 3 && 17u - 55 <= ++x % y
- (2 > 3) && (17u - 55 <= ++x % y)
- false && (17u - 55 <= ++x % y)
- false

Exercise

- Evaluate the following expressions by hand and write down all intermediate steps. You can assume that x is a variable of type int with value 1.

- Werten Sie die folgenden Ausdrücke von Hand aus und geben Sie dabei alle Zwischenschritte an. Sie können annehmen, dass x eine Variable vom Typ int mit dem Wert 1 ist.

  - x == 1 || 1 / (x - 1) < 1
  - !(1 && x) + 1

Exercise

x == 1 || 1 / (x - 1) < 1

1 == 1 || 1 / (x - 1) < 1

true || 1 / (x - 1) < 1

true

Exercise

```
!(1 && x) + 1
!(true && x) + 1
!(true && 1) + 1
!(true && true) + 1
!true + 1
false + 1
0 + 1
1
```

Agenda

- HW #1 feedback
- Expressions
- **if Statements**
- for loop
- const
- assert
- HW#3 hints

if statement

```
if (condition) {
    statement;
}
```

```
if (a==5) {
    cout << "a is equal to 5!\n";
}
```

- If „condition“ is TRUE, the instruction(s) inside the code block { } are executed.
- In contrary, if „condition“ is FALSE, the block is skipped.
- In case of a single instruction, the { } can be omitted.

```
if (condition)
    statement;
```

if-else statement

```
if (condition) {
    statement;
}
else
{
   otherStatement;
}
```

```
if (a==5) {
    cout << "a is equal to 5!\n";
}
else {
    cout << "a is not equal to 5!
    \n";
}
```

- The „else“ code block is executed if all other conditions are FALSE.

if statement

```
if(firstCondition) {
        DoSomething();
}
else if(secondCondition) {
        DoSomethingElse();
}
else {
        IfEverythingElseFails();
}
//”the bottom”
```

- else-if is only checked if the preceding condition(s) are FALSE.
- Many „if - else if – else if – else if...“ statements can be chained to form a complex program flow.

Example - leap year

```
#include <iostream>
int main()
{
  unsigned int year;   std::cin >> year;

  if ( year % 4 ) std::cout << "commony year\n";
  else if ( year % 100 ) std::cout << “leap year\n";
  else if ( year % 400 ) std::cout << "commony year\n";
  else std::cout << "leap year\n";

  return 0;
}
```

Example - compares two integers

```
1   #include <iostream>
2   int main()
3   {
4      unsigned int n;
5      std::cin >> n;
6      unsigned int m;
7      std::cin >> m;
8
9
10
11
12
13
14
15     return 0;
16  }
```

Example - compares two integers

```
1   #include <iostream>
2   int main()
3   {
4      unsigned int n;
5      std::cin >> n;
6      unsigned int m;
7      std::cin >> m;
8
9
10     if (m > n)
11         std::cout << m << " is bigger than " << n << "\n";
12     else
13         std::cout << n << " is bigger or equal to " << m << "\n";
14
15     return 0;
16  }
```

Agenda

- HW #1 feedback
- Expressions
- if Statements
- **for loop**
- const
- assert
- HW#3 hints

for loops structure

```
for (init-statement condition; expression) {
    statement
}
```

- example for `init-statement`: `int i = 0;`
- example for `condition`: `i < 9`
- example for `expression`: `++i`
- example for `statement`: `std::cout << i << std::endl;`
- full example:

```
for (int i = 0; i < 9; ++i) {
    std::cout << i << std::endl;
}
```

for loops hints

```
for (init-statement condition; expression) {
    statement
}
```

- The expression is executed after the statement
- empty condition is true

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
sum:    0
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
sum:    0
i:      1
```

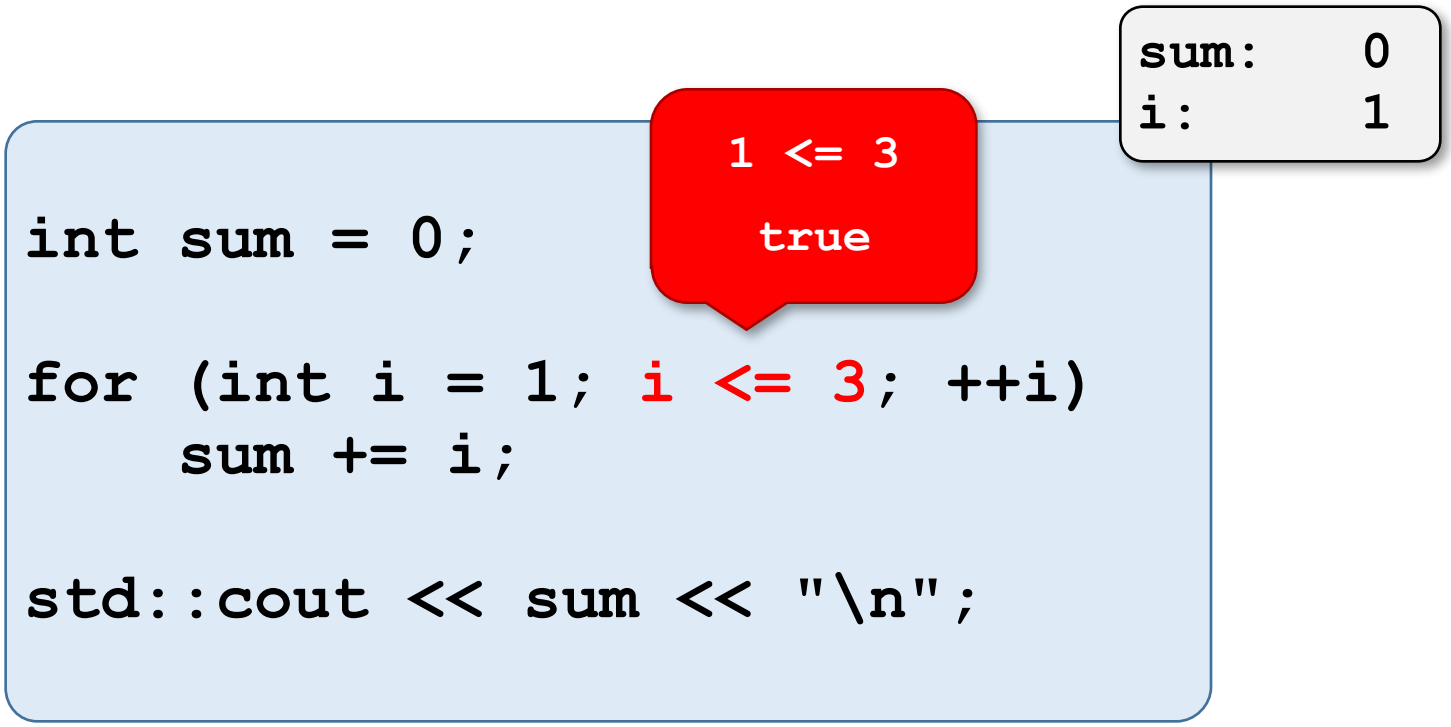
```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
sum:    0
i:      1
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop



Example – `for` Loop

```
sum:    1
i:      1
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
sum:    1
i:      2
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
sum:    1
i:      2
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

2 <= 3

true

```
sum:    1
i:      2
```

Example – `for` Loop

```
sum:    3
i:      2
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    3
i:      3
```

Example – `for` Loop

```
sum:    3
i:      3
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

3 <= 3

true

```
sum:    3
i:      3
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
i:      3
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

```
sum:    6
i:      4
```

Example – `for` Loop

```
sum:    6
i:      4
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Example – `for` Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

4 <= 3

false

```
sum:    6
i:      4
```

Example – `for` Loop

```
sum:    6
```

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

Exercise

Write a program strangesum.cpp that:

1. Reads a number n > 0 from standard input

2. Outputs the sum of all positive numbers up to n that are **odd** but **not divisible by 5**.

Exercise

```cpp
// Program: strangesum.cpp
#include <iostream>
int main()
{
  // input
  unsigned int strangesum = 0;
  unsigned int n;
  std::cin >> n;

  // computation




  // output
  std::cout << "The strange sum is " << strangesum << ".\n";

  return 0;
```

Exercise

```
// Program: strangesum.cpp
#include <iostream>
int main()
{
  // input
  unsigned int strangesum = 0;
  unsigned int n;
  std::cin >> n;

  // computation
  for (unsigned int i = 1; i <= n; i++)
    if (i%2 == 1)
      if (i%5 != 0)
        strangesum += i;

  // output
  std::cout << "The strange sum is " << strangesum << ".\n";

  return 0;
}
```

for loops tips

- infinite loops
- wrong halting condition
- missing brackets { }

Agenda

- HW #1 feedback
- Expressions
- if Statements
- for loop
- **const**
- assert
- HW#3 hints

Const-Guideline

Before you declare a variable, think about whether its value will be changed later or not!

If not, use the keyword `const` to declare the variable as constant.

`const` – Necessary?

- Protects against unintended changes

`const` – Necessary?

- Protects against unintended changes
  - Compiler error message

```
rewrite_const.cpp: In function 'int main()':
rewrite_const.cpp:8:4: error: assignment of read-only variable 'i'
  i = 4;
    ^
make: *** [rewrite_const] Error 1
```

`const` – Necessary?

- Protects against unintended changes
  - Compiler error message

```
rewrite_const.cpp: In function 'int main()':
rewrite_const.cpp:8:4: error: assignment of read-only variable 'i'
  i = 4;
    ^
make: *** [rewrite_const] Error 1
```

- Communicate to reader
  - Reader knows: value will not change

Exercise

Make this `const`-correct.

1. Program:

```
#include <iostream>
int main ()
{
    const int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```


(From: Script Exercise 5)

Exercise

Problem:

input operator >> changes constant variable

1. Program:

```
#include <iostream>
int main ()
{
    const int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```

Solution:

```
#include <iostream>
int main ()
{
    int a = 5;
    std::cin >> a;
    std::cout << a + 5;

    return 0;
}
```


(From: Script Exercise 5)

Exercise

Make this `const`-correct.

2. Program:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    int c = 2*b;
    b = b*b;

    return 0;
}
```


(From: Script Exercise 5)

Exercise

Problem:

- `c` should be `const`.
- `c` is initialized without a later use.

2. Program:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    int c = 2*b;
    b = b*b;

    return 0;
}
```

Solution:

```
int main ()
{
    const int a = 5;
    int b = 2*a;
    const int c = 2*b;
    b = b*b;

    return 0;
}
```


(From: Script Exercise 5)

Exercise

Make this `const`-correct.

3. Program:

```
int main ()
{
    const int a = 5;
    a = 5;

    return 0;
}
```


(From: Script Exercise 5)

Exercise

Problem:

`a = 5;` overwrites `a` with **same** value.

But `a` is `const`; `const` prevails.

3. Program:

```
int main ()
{
    const int a = 5;
    a = 5;

    return 0;
}
```

Solution:

Remove `const` or

`a = 5;`

(From: Script Exercise 5)

Agenda

- HW #1 feedback
- Expressions
- if Statements
- for loop
- const
- **assert**
- HW#3 hints

Errors in Code…

- Problem:
  We want to avoid certain values for a variable.

- Question:
  How?

General Hints

- `assert(expr);`
  - `expr` is `true`: nothing happens
  - `expr` is `false`: stop program

Example

Problem: Some inputs are dangerous.

```
#include <iostream>

int main () {
    int a;
    int b;
    std::cin >> a >> b;

    // Output: a/b
    std::cout << a/b << "\n";

    return 0;
}
```

Problem for:

`b == 0`

Example

Problem: Some inputs are dangerous.

`assert` ensures

`b != 0`

```
}
```

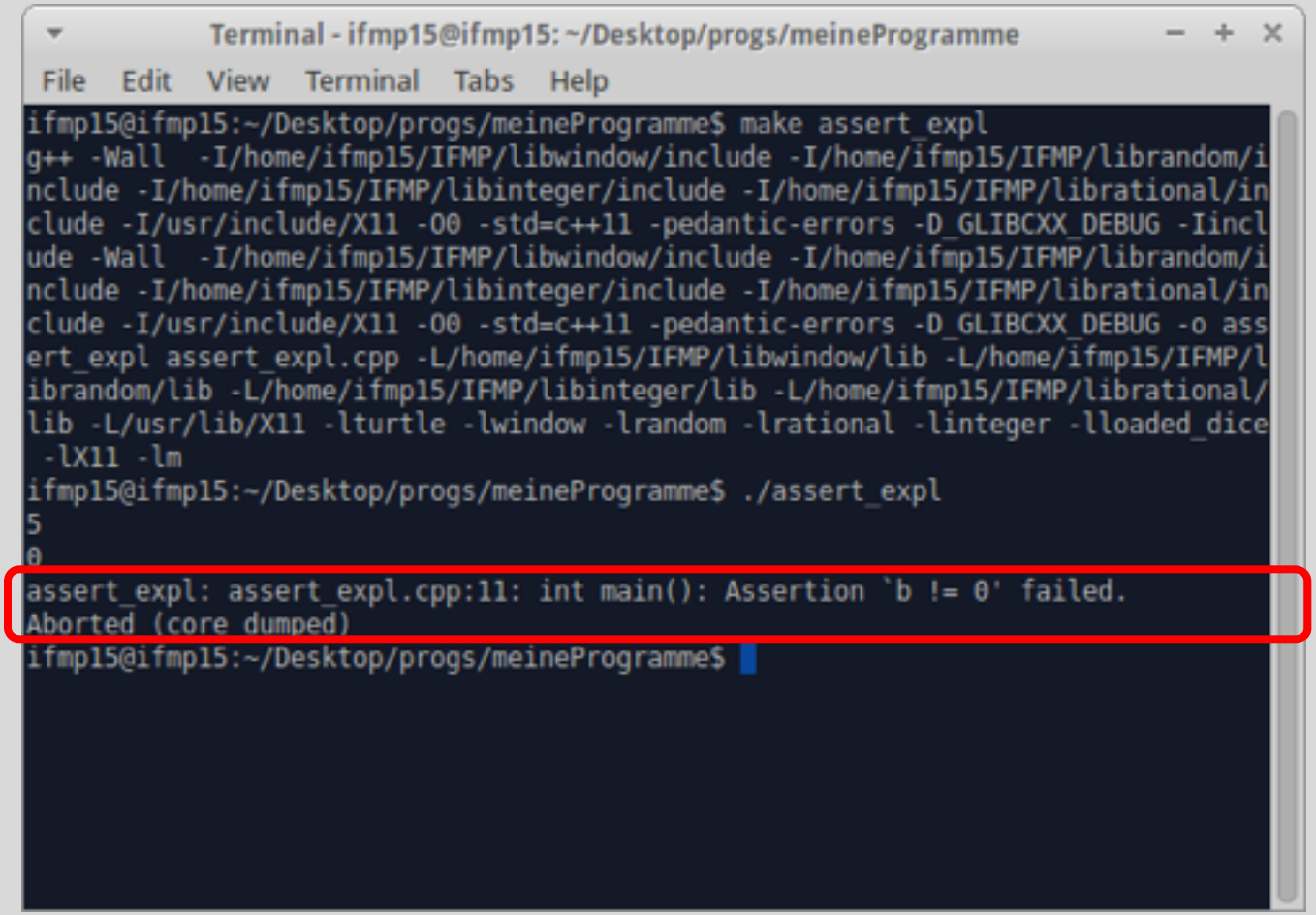
Example

Problem: Some inputs are dangerous.

```
#include <iostream>
#include <cassert>

int main () {
    int a;
    int b;
    std::cin >> a >> b;
    assert(b != 0);

    // Output: a/b
    std::cout << a/b << "\n";

    return 0;
}
```

Example

Pro


Terminal - ifmp15@ifmp15: ~/Desktop/progs/meineProgramme
File Edit View Terminal Tabs Help
ifmp15@ifmp15:~/Desktop/progs/meineProgramme$ make assert_expl
g++ -Wall -I/home/ifmp15/IFMP/libwindow/include -I/home/ifmp15/IFMP/librandom/i
nclude -I/home/ifmp15/IFMP/libinteger/include -I/home/ifmp15/IFMP/librational/in
clude -I/usr/include/X11 -O0 -std=c++11 -pedantic-errors -D_GLIBCXX_DEBUG -Iincl
ude -Wall -I/home/ifmp15/IFMP/libwindow/include -I/home/ifmp15/IFMP/librandom/i
nclude -I/home/ifmp15/IFMP/libinteger/include -I/home/ifmp15/IFMP/librational/in
clude -I/usr/include/X11 -O0 -std=c++11 -pedantic-errors -D_GLIBCXX_DEBUG -o ass
ert_expl assert_expl.cpp -L/home/ifmp15/IFMP/libwindow/lib -L/home/ifmp15/IFMP/l
ibrandom/lib -L/home/ifmp15/IFMP/libinteger/lib -L/home/ifmp15/IFMP/librational/
lib -L/usr/lib/X11 -lturtle -lwindow -lrandom -lrational -linteger -lloaded_dice
-lX11 -lm
ifmp15@ifmp15:~/Desktop/progs/meineProgramme$ ./assert_expl
5
0
assert_expl: assert_expl.cpp:11: int main(): Assertion `b != 0' failed.
Aborted (core dumped)
ifmp15@ifmp15:~/Desktop/progs/meineProgramme$

`assert` – Why?

- Still an easy example…
- So **why** and **where** is `assert` useful?

`assert` – Why?

- Still an easy example…

- So **why** and **where** is `assert` useful?
  - **Long programs:** for overview
  - **User-Inputs required:** for safety
  - **Multiple programmers:** for safety

Agenda

- HW #1 feedback
- Expressions
- if Statements
- for loop
- const
- assert
- **HW#3 hints**

HW #3 Hints

- Write a program dec2bin.cpp that inputs a natural number n (including 0) and outputs the binary digits of n in reverse order.

$$\begin{aligned} 12 &\ \% \ 2 = 0 \\ 6 &\ \% \ 2 = 0 \\ 3 &\ \% \ 2 = 1 \\ 1 &\ \% \ 2 = 1 \end{aligned} \qquad \begin{aligned} 12 &\ / \ 2 = 6 \\ 6 &\ / \ 2 = 3 \\ 3 &\ / \ 2 = 1 \\ 1 &\ / \ 2 = 0 \end{aligned}$$

Binary representation of 12!

```
if reading from the bottom up
```