# Exercise Session

**Week 03**

Adel Gavranović
agavranovic@student.ethz.ch

## Today's topics

‣ polybox for session material

‣ Mail to TA

Intro

Expressions

for-loops

Debugging

## Follow-up

- Are **all of you** able to use [code]expert now?
- Use the ”Playground” on [code]expert to test out ideas and play around with stuff you’ve learned. You can find it under ”Code Examples” at the very bottom of the page
- The moodle-page for the Self-Assessments is now open and much better visible from the course page, give them a try!

## Questions re: Homework?

## Expressions

- Repetition: what was a `bool` again?
- Precedences
- (Parenthesis) are your best friends
- Order matters
- Be aware of *short circuits*

## Booleans

- usually just called `bools`
- either `true` or `false`
- `false == 0`
- `true != 0`
- whenever `true` turns into a number (`int`), it'll be the number `1`
- whenever a number that is `!= 0` turns into a `bool`, it'll turn into `true`

## Precedences Ranking

1. `a++, a--`
2. `++a, --a, -a, !a, *a, &a`
3. `*, /, %`
4. `+, -`
5. `<, <=, >, >=`
6. `== !=`
7. `&&`
8. `||`
9. `=, +=, -=, *=, /=, %=`

## (use) (parenthesis) (!)

- (parenthesis) work much like in real math
- used to make the correct evaluation obvious
- or to change the way the expression is evaluated

**Task**

Make the evaluation of the following expression obvious:

```
3 < 4 + 1 && 2 < 3
```

Hint: use the previous slide

**Solution**

```
(3 < (4 + 1)) && (2 < 3)
```

## Multiple operators with same precedence

**Quick Task**

How would you parenthesize the expression below to make it obvious?

```
false && false && true
```

**Quick Solution**

Just ”read” from left to right:

```
(false && false) && true
```

## Short Circuits

**Short Circuit**

"`&&`" and "`||`" evaluate the expression to their left first and if the compiler (the machine) can imply the correct evaluation already, it *won't check the right side*.

What are the implications of that? See next slide.

## Short Circuit in Code

```
if (3 > 2 && 10 > 11){
   std::cout << "Of course not!\n";
}  // not a short circuit evaluation

int a = 3;

if (false && ++a < 2){
   std::cout << "Of course not!\n";
} // a short circuit evaluation

std::cout << a << "\n"; // what will be the output?

if (++a < 2 && false){
   std::cout << "Of course not!\n";
} // another short circuit evaluation

std::cout << a << "\n"; // what will be the output?
```

## Let’s check comprehension

### Task

Evaluate the following expression by hand and write down each intermediate step. Assume `int x = 1`:

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

Remember: Parenthesis are your friends.

### Solution

First: parenthesize!

`(x == 1) || ((1 / (x - 1)) < 1)`, start on left side

```
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1)*
true
```

*(`true || whatever`) always eval’s to (`true`)

## And another one

**Task**

Evaluate the following expression by hand and write down each intermediate step. Assume `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

**Solution**

```
!(1 < 2 && x == 1) + 1
(!((1 < 2) && (x == 1))) + 1
(!((true) && (true))) + 1
(!(true)) + 1
false + 1
0 + 1
1
```

## Questions?

## for-loops

- what is a *loop*
- we'll get to know *scopes* a bit better
- an intro to *Program Tracing*

## Scope

- Any time you use the {squiggly brackets} you create a *scope*
- You can think of a scope as a closed world in itself
- Information can’t flow out of the scope, but outside information (variables etc.) is availabe inside the scope
- When the scope closes (program hits the right ”}”) the information inside of that scope dies
- ([code]expert example)

## General structure of a for-loop

```
for(init; contition; expression){
   statement 1;
   statement 2;
   ...
}
```

### Important note

the `expression`-part will get executed **after** the `statements`.

## Program Tracing

"*Program Tracing* is the process of executing program code by hand, with concrete inputs."

It's quite an important skill in the beginning. At some point, you'll be able to do it in your head. You'll see an example with a simple for-loop in the next few slides.

## Concrete example of a for-loop

:: open ”example of a for-loop”-slides ::

## Questions?

## Exercise: Strange Sum

**Task**

Open ”Strange Sum” in your [code]expert and give it a try it yourself. Solve it individually with pen and paper.

**Description:**
Write a program that reads a number $n > 0$ from standard input and outputs the sum of all positive numbers up to $n$ that are odd but not divisible by 5. (10min)

## Space for student solution (attempts)

## Solution to ”Strange Sum”

```
// 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){
            strangesum += i;
      }
   }
}

// output
std::cout << strangesum << "\n";
```

## Sweeter solution to ”Strange Sum”

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i++){
   if( ((i % 2) == 1) && (i % 5) ){
           strangesum += i;
   }
}

// output
std::cout << strangesum << "\n";
```

## Even sweeter solution to ”Strange Sum”

```cpp
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

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

// output
std::cout << strangesum << "\n";
```

## Questions?

## Exercise: Largest Power

**Task**

Open ”Largest Power” in your [code]expert and give it a try it yourself. Solve it individually with pen and paper.

**Description:**
Write a program that inputs a positive natural number *n* and outputs the largest number *p* that is a power of 2 and smaller or equal to *n*. (15min)

**Task**

Now, discuss with your neighbor. Did they have a similar approach? What can you learn from each other? (7min)

## Space for student solution (attempts)

## Solution to ”Largest Power”

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

int main () {
  unsigned int n;
  std::cin >> n;
  assert(n >= 1);

  unsigned int power = 1;
  for (; power <= n / 2; power *= 2);

  std::cout << power << std::endl;

  return 0;
}
```

## Debugging

"*Debugging* is the process of finding and resolving bugs (defects or problems that prevent correct operation) within programs, software, or systems."

You'll spend a lot of time doing this, so try to do it effectively.

**Task**

Propose a way of finding the bug in the following code.

## Debugging non_terminating.cpp

```
int main () {
    const int n = 6;

    // Compute n^12
    int prod = 1;
    for (int i = 1; 1 <= i < 13; ++i) {
        prod *= n;
    }

    // Output stars
    for (int i = 1; i < prod; ++i) {
        std::cout << "*";
    }
    std::cout << "\n";
    return 0;
}
```

## Live demo

**Disclaimer:** This might go horribly wrong

## Debugging non_terminating.cpp

**Question**

How can we know at which line the program gets stuck?

**Answer**

Try to print something to the console at various lines and see what gets printed.

**Question**

Why doesn't the first loop terminate?

**Answer**

The condition is wrong. It should be: `1 <= i && i < 13`.

## Debugging non_terminating.cpp

**Question**

How can we investigate further why the program does not print anything?

**Answer**

Print the value of `prod` after the first loop

**Question**

How we can get to know why `prod` became negative?

**Answer**

Print the value of `prod` in *each* interation of the first loop

## Final Questions?