Exercise Session
Week 04

Adel Gavranović
`agavranovic@student.ethz.ch`

Today's topics

▸ polybox for session material

▸ Mail to TA

Introduction

Expressions

Loops

Computing Mathematical Series

Scopes

Comments on last Exercise Session

- I'm sorry the live demo didn't work out. I think I messed up by not compiling the "improved" code properly.

Comments on last `[code]expert` Exercises

- please don't use `using namespace std`
- don't use libraries or features that we didn't cover in the lectures (yet)
- read the task carefully
- deadlines are strict regarding XP
- submit your code, no matter how "bad" it seems: you can learn a lot from doing so!
- use comments and tabs
- try to structure your answers. It looks nicer and makes grading easier
- when solving exercises, you're **allowed** to use everything (handouts, slides, summaries(!), recordings) we give you

Questions or Comments re: Exercises?

Objectives Checklist

After the exercise session, take a look at this slide again and make sure you can tick the boxes, and if you can't:
Ask questions or send me an e-mail. I'm here to help!

Now I...

☐ can evaluate complex expressions involving arithmetic and boolian operators

☐ can encode mathematical sums into `C++`

☐ know about the types `float` and `double` in `C++` (much more on them soon)

☐ can implement `for`, `while` and `do-while`-loops

☐ can trace programs that have `for`, `while` and `do-while`-loops in them

☐ can turn each kind of loop into a different kind of loop

Quick Recap on Types

Types (we've covered so far)

- logical variables: bool {false, true}
- integers: unsigned int, int {-7, 2, 0}
- floating point numbers: float, double {1.4, -4.3, 7.0}

Sometimes there are multiple types in one (expression).
How do we compare different types with each other?

Generality Order of Types (we've covered so far)

bool < int < unsigned int < float < double
Types always convert to the most general type in any given
expression.

A Way to think about Types

Type (literal)   Approximates

bool $\{\text{false}, \text{true}\}$
unsigned int (u) $\{\mathbb{N}\}$
int $\{\mathbb{Z}\}$
float (f) $\{\mathbb{R}\}$
double $\{\mathbb{R}\}$, but *double* the precision

Evaluating Types

```
std::cout << 5.0/2 << std::endl;
// what type and value will this return and why?
```

Solution

`double`, `2.5`, because the compiler will convert the `int 2` into a `double 2.0`, in order to calculate this expression.

```
std::cout << (1/2)*5.0/2 << std::endl;
// what type and value will this return and why?
```

Solution

`double`, `0`, because the compiler will first calculate the expression on the left `1/2` which evaluates to `0` because it's an integer division. The rest is trivial, because `0*anything` evaluates to `0`. But that `0` will have type `double`.

Literals

Literals

There are certain letters the compiler associates with certain
types. So, if you want to tell the Compiler *"Hey, don't treat this
2.0 as a double, but instead as a float"* you'd have to add f at
the end of the value. Like this:

```
std::cout << (5/2)*5.0f/2 << std::endl;
// what type and value will this return and why?
```

Solution

float, 5.0, (which can be written as 5.0f).
First, the compiler evaluates 5/2, which results in 2, because
integer division works that way. Then the compiler calculates
2.0f*5.0f: The int 2 has been turned into a float 2
because float is the more general of the types that are
involved. The same for the *2 later.

Questions?

Exercises

:: exercises_slides.pdf ::

(Solutions can be found on exercises_handout.pdf)

Exercise in Loops and Tracing

```cpp
std::cout << "Enter a number: ";
unsigned int n;
std::cin >> n;
// Can a user observe a difference between the outputs?
// loop 1
for (unsigned int i = 1; i <= n; ++i) {
  std::cout << i << "\n";
}
// loop 2
unsigned int i = 0;
while (i < n) {
  std::cout << ++i << "\n";
}
// loop 3
i = 1;
do {
  std::cout << i++ << "\n";
} while (i <= n);
```

(if n = largest int value)

will continue
printing forever

0110
0111
1000
⋮
┌─────┐
│1111 │
└─────┘
┌─────┐
1│0000│ → 0
└─────┘
↗

Program Tracing

We've covered Program Tracing last week, but here's an extensive (and better) guide on how to do it: (▸ Link)

Converting Loops (`for` → `while`)

```
// TASK: Convert the following for-loop
// into an equivalent while-loop:

for (int i = 0; i < n; ++i) {
 BODY          ← nat  actuall  cpp-code
}
```

Converting Loops (`for` → `while`**)**

```
// TASK: Convert the following for-loop
// into an equivalent while-loop:

for (int i = 0; i < n; ++i) {
 BODY
}
```

```
// SOLUTION:

int i = 0;

while(i < n){
 BODY
 ++i;
}
```

Converting Loops (while → for)

```
// TASK: Convert the following while-loop
// into an equivalent for-loop:

while(condition){
 BODY
}
```

```
// SOLUTION:

for(;condition;){
 BODY
}
```

Converting Loops (`do-while` → `for`)

```
// TASK: Convert the following do-while-loop
// into an equivalent for-loop:

do{
 BODY
}while(condition)
```
```
// SOLUTION:

BODY

for(;condition;){
 BODY
}
```

Introduction
○○○○

Expressions
○○○○○○

Loops
○○○○○●

Computing Mathematical Series
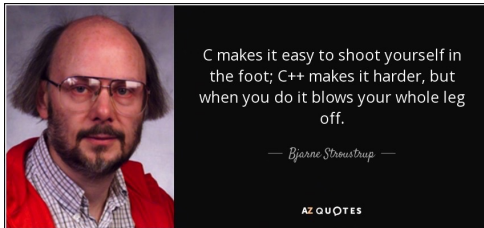○○

Scopes
○

Questions?

shifting indizes is possible, just look
out for "off-by-one" errors

`https://en.wikipedia.org/wiki/Off-by-one_error`

→ will happen often when
"iterating" over arrays/vectors

(more on that later in the course)

Daddy of
C++
(obviously) }

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.

— *Bjarne Stroustrup* —

AZ QUOTES

From Sum to Loop

Mathematical sums can be turned into programming loops.

Math:

$$\sum_{i=0}^{n} f(i)$$

C++ :

```cpp
int n = 0;
int sum = 0;

for(int i = 0; i <= n; i++){
  sum += f(i);
}
```

From Mathematical Series to Loops

Taylor Series on [code]expert

Write a program that computes $\sin x$ rounded to six significant digits. Hint: Think which loop you should use.

Hint: Use the MacLaurin Series.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

numerator: $(-1)^n x^{2n+1} \{ \cdots$

denominator: $(2n+1)! \{ \cdots$

Hint: "split up" total_sum from each **term**

Task

as long as needed

- For 10 minutes, think about how you would solve it by using only pen and paper
- Now get together with your desk neighbor(s) and try to implement it on [code]expert ~~for 10 minutes~~ during rest of session

Scopes

see exercises_handout.pdf

If there still are questions after reading through it, feel free to
write me an e-mail or ask in the next exercise session