

Session 03

Control Flow II, Floating Point I

Informatik I

Lan Zhang

07.03.2025



Schedule

1. Review of Last Week
2. Main Topics
 - Scopes
 - Arithmetic Expressions
 - Loops
3. In-Class Code Examples
 - Taylor Series

Scope = Validity range of a declaration

- Local variables only exist within their scope of definition {...}^{*} and in all nested scopes
 - Value change in subscopes also valid outside
 - Value declaration not valid outside

```
int main() {  
    int global = 1;  
    {  
        ++global;  
        int local = global;  
    }  
    std::cout << global; // 2  
    std::cout << local; // error: undeclared variable  
    return 0;  
}
```

} For exact reference: [Scopes - C++ Guides](#)

^{*}loops, functions etc.

Floating Point Number Systems

Defined by 4 natural numbers: $F(b, p, e_{min}, e_{max})$

- $b \geq 2$: base
- $p \geq 1$: precision (= number of digits, Mantisse)
- e_{min} : smallest exponent
- e_{max} : largest exponent

$$\pm d_0.d_1\dots d_{p-1} \cdot \beta^e = F^*(\beta, p, e_{min}, e_{max})$$

- Eine Ziffer vor dem Komma, der Rest dahinter
- Normalisierte Darstellung ist eindeutig
- Die Zahl 0 und alle Zahlen kleiner als $\beta^{e_{min}}$ haben keine normalisierte Darstellung

Wert	Formel
Anzahl positiver Werte	$(b - 1) \cdot b^{p-1} \cdot (e_{max} - e_{min} + 1)$
Anzahl Werte	$2 \cdot (b - 1) \cdot b^{p-1} \cdot (e_{max} - e_{min} + 1) = 2 \cdot (\# \text{ positiver Werte})$
Grösste Zahl	$(b^p - 1) \cdot b^{e_{max}-p+1}$
Kleinste positive Zahl	$b^{e_{min}}$

Floating Point Numbers

Double

- F(2,53,-1022,1023) 64 Bits: 1 Bit for sign, 52 Bits for precision, 11 Bits for exponent
- default type for floating-point literals

Float

- F(2,24,-126,127) 32 Bits: 1 Bit for sign, 23 Bits for precision, 8 Bits for exponent

Explicit Typecasting: (type)value

E.g.: (double) 1.27f, (float) 1.05

Note: 0.1f \neq 0.1d -> if not exactly representable as binary number, different precisions due to rounding

Arithmetic Expressions

- Different data types can be used in the same expression, but they are **implicitly converted**
- Similar to the operators, different data types in C++ also have precedencies that you need to be aware of.

Evaluation Order:

`bool < int < unsigned int < float < double`

Least general  Most general (representative)

For example:

`5 / 2 == 2` (int)

`5 / 2.0 == 2.5` (int < double -> double)

`3.0 + 5 / 2 == 5` (precedence of operators)

`3.0 + true / 2.0 * 5 - 6.0f / 3 == 3.5` (double + bool / double * int (double) – float / int (float) -> double)

Arithmetic Expressions: Examples

1. Validity
2. L-value or R-value
3. Result

```
int x = 1;  
int y = -1;
```

A. $(y++ < 0 \ \&\& \ y < 0) + 2.0$

Arithmetic Expressions: Examples

1. Validity
2. L-value or R-value
3. Result

```
int x = 1;  
int y = -1;
```

B. $y = (x++ = 3)$

Arithmetic Expressions: Examples

1. Validity
2. L-value or R-value
3. Result

```
int x = 1;  
int y = -1;
```

C. $3.0 + 3 - 4 + 5$

Arithmetic Expressions: Examples

1. Validity
2. L-value or R-value
3. Result

```
int x = 1;  
int y = -1;
```

D. `5 % 4 * 3.0 + true * x++`

If, else if, else instructions

```
if (condition) {  
    statement;  
} \\ statement is executed if condition is true  
else if(condition){  
    statement;  
} \\ like if-clause but only looked at if condition before  
false  
else {  
    statement;  
} \\ executed if other conditions are false
```

Switch instruction

- If a case is not ended with a break, the underlying cases are still executed until a break is reached.

```
switch(expression) {  
    case 1*: statement1; \\ matching case is executed  
    case 2: statement2; break; \\ all statements until break are executed  
    case 3: statement3;  
    case ...;  
    break;  
    default: statement else; \\ if no case matches  
}
```

*: case arguments can only be constant literals

Iterations

while-loop:

loops over a block of code until condition is no longer satisfied

```
while (condition) {  
    statement;  
    ...  
} \\ statement is executed while condition == true
```

do-while-loop:

executes block of code, then checks condition

```
do {  
    statement;  
    ...  
} while (condition); \\ statement is executed once, afterwards while-  
loop
```

Loop Correctness

Observe the difference between the outputs produced by the three loops

```
int n; std::cin >> n;
int i;
// loop 1
for (i = 1; i <= n; ++i) {
    std::cout << i << "\n";
}
// loop 2
i = 0;
while (i < n) {
    std::cout << ++i << "\n";
}
// loop 3
i = 1;
do {
    std::cout << i++ << "\n";
} while (i <= n);
```

Loop Conversion

for-loop -> while-loop:

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

while-loop -> for-loop:

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

do-while-loop -> for-loop, while-loop:

```
do {  
    statement  
} while (condition);
```


Loop Conversion

for-loop -> while-loop:

```
for (int i = 0; i < n; ++i) {  
    statement;  
  
    if(i==0) continue;  
  
    // not the same effect as i  
    is increased before next iteration  
}
```

In-Class Code Examples

Computing Series

Consider the formula:

$$\frac{1}{n!}$$

How would you represent this as a series?

$$\prod_{k=1}^n \frac{1}{k}$$

What do we need?

- Input variable
- Loop variable
- Result

What is the terminating condition and how does the result evolve?

In-Class Code Examples

Taylor Series

Write a program that computes $\sin(x)$ with precision $1e-6$.

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

How can you derive results for $n + 1$ using the results of n ?