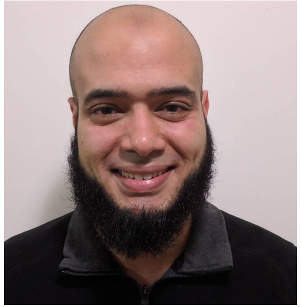
C++ Programming

Precedence

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Remember math rules

- How do you solve: 2 + **3 * 4** - 6/2
    - * applied first so ⇒ 2 + 12 - **6/2**
    - / division applied ⇒ **2 + 12** - 3
    - Then left to right ⇒ 14 - 3 ⇒ 11
    - In general: * / are applied first before - +
- What about 2 + **3 * (4** - 6/2)
    - Inside parentheses first
    - So solve 4-6/2 ⇒ 4-3 ⇒ 1
    - Now: 2 + 3 * 1 ⇒ 2 + 3 ⇒ 5
- Math defines for us the order of operations
    - E.g. () is first. * / are before + -
    - This is called precedence

Operator Precedence

- What about the new other operators in C++?
    - (++ -- - !) before (* / %) before (+ -) before (=   +=   -=   *=   /=   %=)
- ++x - y/z + t--
    - ++x first
    - t--
    - y/z

Operator Precedence: ()

- Use parentheses to force order / resolve ambiguity
- 2 + 3 * (7 - 6)/2   : First (7-6)
  - 2 + **3 * 1** / 2 ⇒ 2 + **3** / 2 ⇒ 2 + 1.5 ⇒ 3.5 (if doable was used, otherwise 3)
- ++x - y/**(z + t--)**
  - first (z + t--)
    - Which needs first t--
  - Then ++x
  - y/<>
  - Then overall

Operator Precedence: ()

- How to solve?
  - Find some deepest parentheses, compute its expression: and so on till no parentheses
- (a + (b - (d * e))) / (a + ++c) + ((1+((x+y)*2)) * z)
  - Let a = 1, b = 2, c = 3, d = 4, e = 5, x = 6, y = 7, z = 1
  - (x+y) ⇒ (a + (b - (d * e))) / (a + ++c) + ((1+**(13*2)**) * z)
  - (13*2) ⇒ (a + (b - (d * e))) / (a + ++c) + (**(1+26)** * z)
  - (1+26) ⇒ (a + (b - (d * e))) / (a + ++c) + **(27 * z)**
  - (27 * z) ⇒ (a + (b - (d * e))) / **(a + ++c)** + 27
  - (a + ++c) ⇒ (a + (b - **(d * e)**)) / 5 + 27 [notice ++c, then c = 4
  - (d * e) ⇒ (a + ++c) ⇒ (a + **(b - 20)**) / 5 + 27
  - (b - 20) ⇒ (a + ++c) ⇒ **(a - 18)** / 5 + 27
  - (a - 18) ⇒ **-17 / 5** + 27 ⇒ 23.6
- What if it was ++b not ++c? No guarantee for the answer! As there is 2 bs in the expression

Operator Associativity

- What if operators have **the same priority**? E.g. + -
  - Associativity: group either from left or from right
- Let’s say we have expression: 10 - 6 + 3
- Left-to-right associativity: **group** from left to right
  - **(10 - 6)** + 3 ⇒ 4 + 3 = 7
  - **7-6**+5-4+3-2+1 ⇒ **1+5**-4+3-2+1 ⇒ **6-4**+3-2+1 ⇒ **2+3**-2+1 ⇒**5-2**+1 ⇒**3+1** ⇒4
- Right-to-Left associativity: **group** from right to left
  - 10 - **(6 + 3)** ⇒ 10 - 9 = 1 [wrong!]
  - 7-6+5-4+3-**2+1** ⇒ 7-6+5-4+**3-3** ⇒ 7-6+5-**4+0** ⇒ 7-6+**5-4** ⇒ 7-**6+1** ⇒ **7-7** ⇒ 0

Operator Associativity

- Left-to-right: * / % + -
- Right-to-left: = += -= *= /= %=
- int x = 10, y = 20, z = 3;
    - x += y += --z *= 9-3-1; [take a moment and try to guess]
    - Highest priority: **--z** so now z = 2
    - Next highest priority **- -** ⇒ 9-3-1 ⇒ 5 (same priority, left to right grouping)
    - Now expression is like
        - x += y += z *= 5; where z=2
        - **Equal** priority (+= += *=) with right to left grouping.
            - z *= 5 ⇒ z = 10 ⇒ x += y += 10
            - y += 10 ⇒ y = 30 ⇒ x += 30 ⇒ x = 40
        - So overall: x=40, y=30, z=10 ….. And don’t code this way :)

Order of evaluation

- This is a bit tricky and many programmers mix it with associativity
- Let’s compute X + Y
  - Assume X and Y are 2 expressions
  - Once the 2 expressions are computed (operand values), eventually X will be added to Y
  - But which expression is **evaluated first**? We don’t know
  - The order of evaluation of operands of individual operators is unspecified
    - It could be X then Y Or Y then X
  - Where is the problem? **side-effect** (a change in state from one expression)
    - x + ++x
    - (++a*b) - (a*d)
- cout << i << “ “<< i++; // undefined behavior until C++17
  - Optional Reading: Undefined, unspecified and implementation-defined **behavior**

Precedence vs Associativity vs Order of evaluation

- **Operator precedence** specifies the **order of operations** in expressions that contain more than one operator (e.g. * before +)
- **Associativity** is about how to **group operands** (of equal priority),
  - But first, we need to evaluate operands/subexpressions
- The compiler can **evaluate operands** and other subexpressions in **any order**
- A - B - C
  - Left to right associativity (A - B) - C
    - Let A/B/C be a subexpression to be evaluated independently, e.g. could be 2*x/4
  - Compiler Evaluation: there are **6 ways** to evaluate them
    - ABC, ACB, BAC, BCA, CAB, CBA
  - Be careful from **side effects**

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."