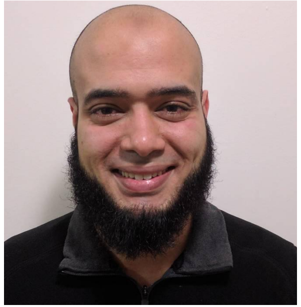
C++ Programming

Fold Expression 1

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

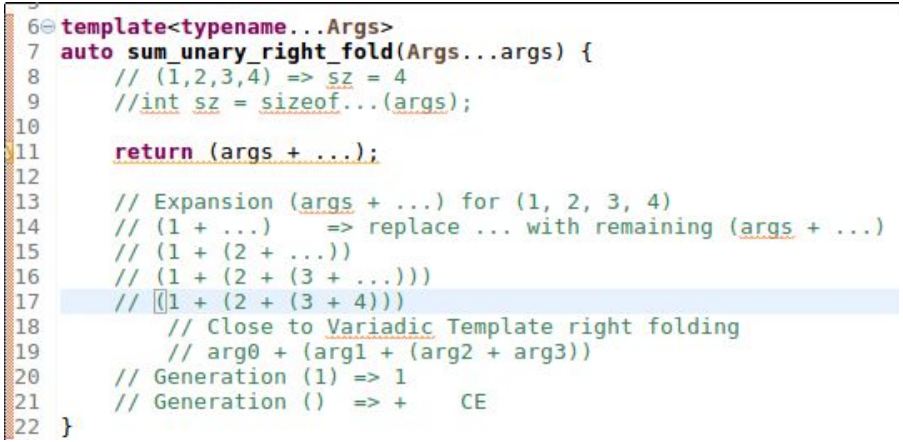
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



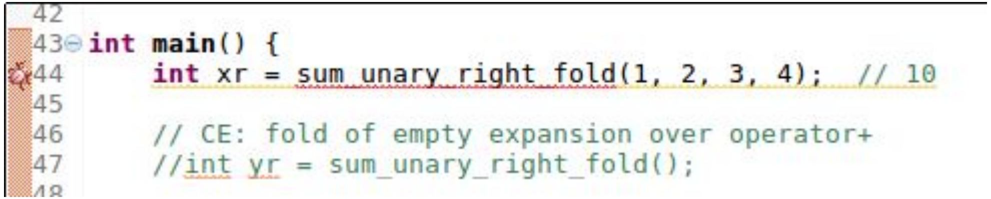
Fold Expression

- Simpler code for **binary** operations over a variable number arguments
- Mainly 4 styles for the folding for the ***compiler to generate***
  - **Unary** right fold, **Unary** left fold
  - **Binary** right fold, **Binary** left fold
- Recall Binary operator: a + b
  - It takes 2 operands
  - The compiler will complaints if generation ended like:
    - 1 + 2 + 3 +
    - + 1 + 2 + 3
    - It must be proper: 1 + 2 + 3

Unary right fold

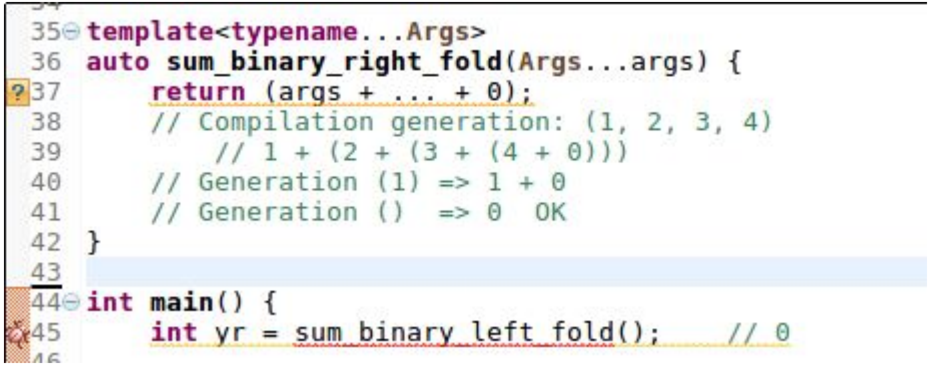


```
template<typename...Args>
auto sum_unary_right_fold(Args...args) {
    // (1,2,3,4) => sz = 4
    //int sz = sizeof...(args);

    return (args + ...);

    // Expansion (args + ...) for (1, 2, 3, 4)
    // (1 + ...)     => replace ... with remaining (args + ...)
    // (1 + (2 + ...))
    // (1 + (2 + (3 + ...)))
    // (1 + (2 + (3 + 4)))
        // Close to Variadic Template right folding
        // arg0 + (arg1 + (arg2 + arg3))
    // Generation (1) => 1
    // Generation ()  => +    CE
}
```



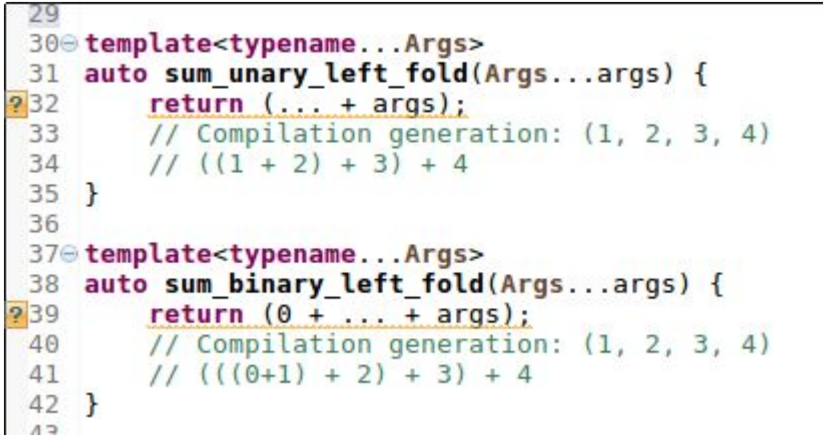
```
int main() {
    int xr = sum_unary_right_fold(1, 2, 3, 4);  // 10

    // CE: fold of empty expansion over operator+
    //int yr = sum_unary_right_fold();
```

Binary right fold



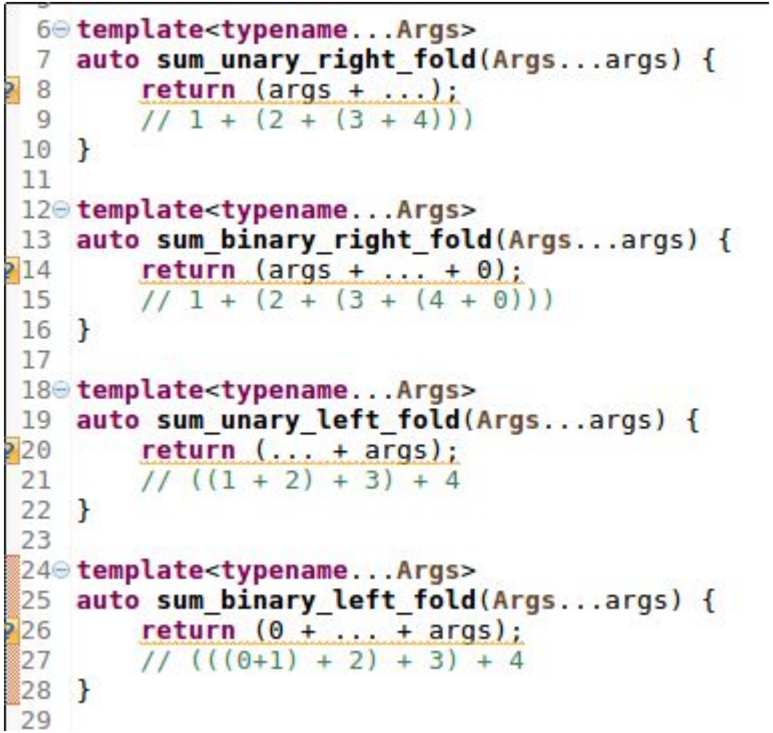
```cpp
template<typename...Args>
auto sum_binary_right_fold(Args...args) {
    return (args + ... + 0);
    // Compilation generation: (1, 2, 3, 4)
        // 1 + (2 + (3 + (4 + 0)))
    // Generation (1) => 1 + 0
    // Generation ()  => 0   OK
}

int main() {
    int yr = sum_binary_left_fold();    // 0
```

Left fold



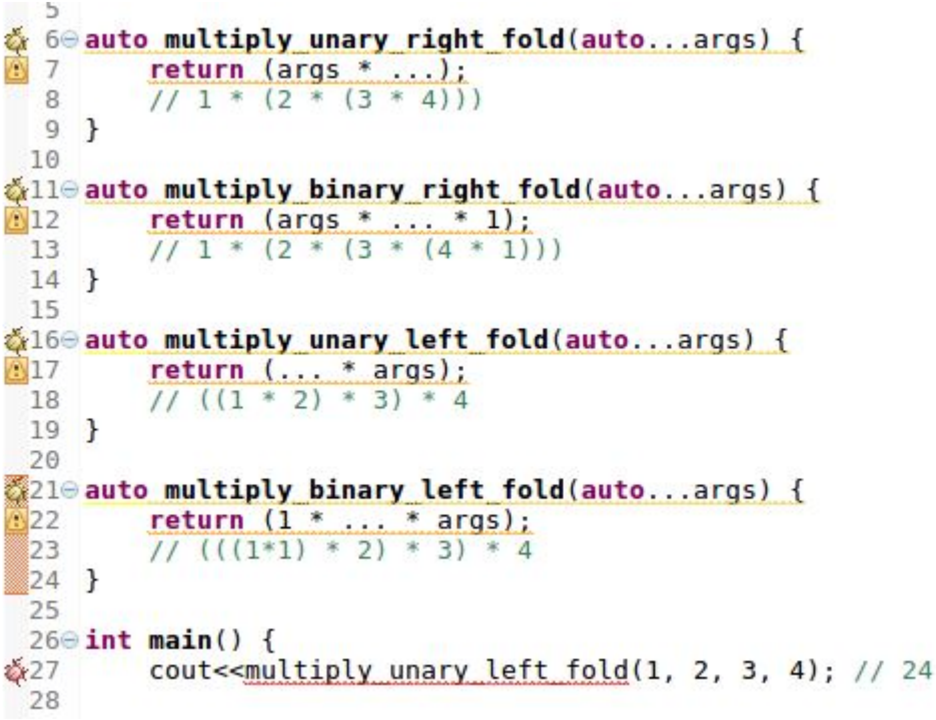
```
template<typename...Args>
auto sum_unary_left_fold(Args...args) {
    return (... + args);
    // Compilation generation: (1, 2, 3, 4)
    // ((1 + 2) + 3) + 4
}

template<typename...Args>
auto sum_binary_left_fold(Args...args) {
    return (0 + ... + args);
    // Compilation generation: (1, 2, 3, 4)
    // (((0+1) + 2) + 3) + 4
}
```

All together



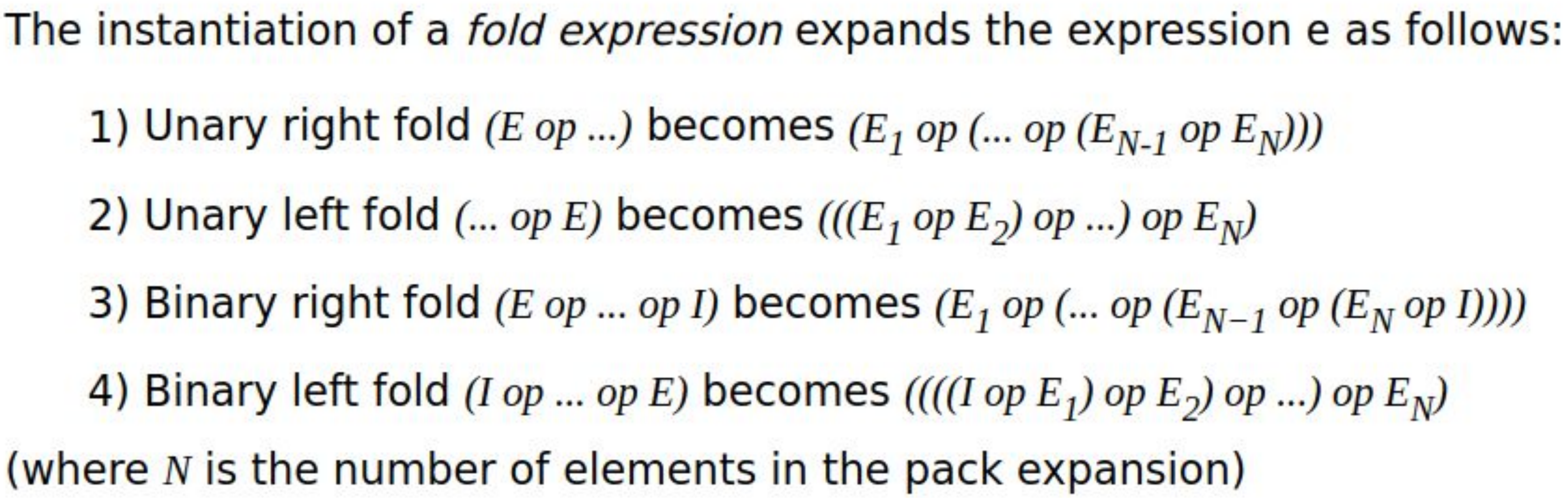
```
template<typename...Args>
auto sum_unary_right_fold(Args...args) {
    return (args + ...);
    // 1 + (2 + (3 + 4)))
}

template<typename...Args>
auto sum_binary_right_fold(Args...args) {
    return (args + ... + 0);
    // 1 + (2 + (3 + (4 + 0)))
}

template<typename...Args>
auto sum_unary_left_fold(Args...args) {
    return (... + args);
    // ((1 + 2) + 3) + 4
}

template<typename...Args>
auto sum_binary_left_fold(Args...args) {
    return (0 + ... + args);
    // (((0+1) + 2) + 3) + 4
}

```

Example: Multiplication + auto



```cpp

auto multiply_unary_right_fold(auto...args) {
    return (args * ...);
    // 1 * (2 * (3 * 4)))
}

auto multiply_binary_right_fold(auto...args) {
    return (args * ... * 1);
    // 1 * (2 * (3 * (4 * 1)))
}

auto multiply_unary_left_fold(auto...args) {
    return (... * args);
    // ((1 * 2) * 3) * 4
}

auto multiply_binary_left_fold(auto...args) {
    return (1 * ... * args);
    // (((1*1) * 2) * 3) * 4
}

int main() {
    cout<<multiply_unary_left_fold(1, 2, 3, 4); // 24

```

Overall 4 cases: Compile time generation



The instantiation of a *fold expression* expands the expression e as follows:

1) Unary right fold $(E \; \mathit{op} \; \ldots)$ becomes $(E_1 \; \mathit{op} \; (\ldots \; \mathit{op} \; (E_{N\text{-}1} \; \mathit{op} \; E_N)))$

2) Unary left fold $(\ldots \; \mathit{op} \; E)$ becomes $(((E_1 \; \mathit{op} \; E_2) \; \mathit{op} \; \ldots) \; \mathit{op} \; E_N)$

3) Binary right fold $(E \; \mathit{op} \; \ldots \; \mathit{op} \; I)$ becomes $(E_1 \; \mathit{op} \; (\ldots \; \mathit{op} \; (E_{N-1} \; \mathit{op} \; (E_N \; \mathit{op} \; I))))$

4) Binary left fold $(I \; \mathit{op} \; \ldots \; \mathit{op} \; E)$ becomes $((((I \; \mathit{op} \; E_1) \; \mathit{op} \; E_2) \; \mathit{op} \; \ldots) \; \mathit{op} \; E_N)$

(where N is the number of elements in the pack expansion)

Supported 32 Operators

- + - * / % ^
- & | = ^= &= |=
- < > == != <= >=
- << >>
- += -= *= /= %=
- <<= >>=
- && || , .* ->*
- Note: In a binary fold, **both** ops must be the same.
    - CE: return (args * ... **+** 1);

Your turn

- Develop the **division** function
- Is left fold is same as right fold? why?

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”