Queens College, CUNY, Department of Computer Science

Object Oriented Programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

July 7, 2018

Prefix and postfix operators

- In this lecture we shall review the **prefix** and **postfix** operators.

1 Prefix and postfix operators

- C++ has specialized operators to **increment/decrement the value of a variable by 1.**
- There are totally four such operators.
- For a variable x of type `char`, `int` , `long`, `float` or `double`, there are four expressions.

pre-increment	`++x`
pre-decrement	`--x`
post-increment	`x++`
post-decrement	`x--`

- The first two operators are called **prefix** operators.
- The last two operators are called **postfix** operators.
- As a matter of fact, we employ such expressions all the time in loops.

```
for (int i = 0; i < n; ++i)
for (int i = 0; i < n; i++)
for (int i = n-1; i >= 0; --i)
for (int i = n-1; i >= 0; i--)
```

- *The key concept to learn here is the meaning of the "pre" and "post" operations.*
- The prefix operation is easy to understand.
- The postfix operation is more complicated.

2 Increment

- We begin with the increment operation.
- Consider the following main program.

```
#include <iostream>

int main()
{
  int n1 = 3;
  int m1 = ++n1;                                          // "pre"
  cout << "prefix:  " << m1 << "   " << n1 << endl;

  int n2 = 3;
  int m2 = n2++;                                          // "post"
  cout << "postfix: " << m2 << "   " << n2 << endl;
  return 0;
}
```

- Both `n1` and `n2` are initialized to the value 3.
- Both `n1` and `n2` are incremented to the value 4.
- However, the values of `m1` and `m2` are **not equal.**
- **Prefix**
  1. The statement `m1 = ++n1` means the following.
     (a) Increment the value of `n1` first.
     (b) Then copy the incremented value to `m1`.
  2. Therefore we first increment `n1` to the value 4.
  3. After that, we copy the value 4 to `m1`.
  4. Therefore `m1 = 4`.
- **Postfix**
  1. The statement `m2 = n2++` means the following.
     (a) **Copy the value of `n2` to `m2` first.**
     (b) **Increment the value of `n2` second.**
  2. *Therefore we first copy the value 3 to* `m2`.
  3. *After the copy, we increment* `n2` *to the value 4.*
  4. **Therefore `m2 = 3`.**
- The program output is the following.

```
prefix:  4   4                    // m1=4  n1=4
postfix: 3   4                    // m2=3  n2=4
```

3 Decrement

- The decrement operators obey the same rules.
- Consider the following main program.

```
#include <iostream>

int main()
{
  int n3 = 3;
  int m3 = --n3;                                          // "pre"
  cout << "prefix:  " << m3 << "   " << n3 << endl;

  int n4 = 3;
  int m4 = n4--;                                          // "post"
  cout << "postfix: " << m4 << "   " << n4 << endl;
  return 0;
}
```

- Both `n3` and `n4` are initialized to the value 3.
- Both `n3` and `n4` are decremented to the value 2.
- However, the values of `m3` and `m4` are **not equal.**
- **Prefix**
  1. The statement `m3 = --n3` means the following.
     (a) Decrement the value of `n3` first.
     (b) Then copy the decremented value to `m3`.
  2. Therefore we first decrement `n3` to the value 2.
  3. After that, we copy the value 2 to `m3`.
  4. Therefore `m3 = 2`.
- **Postfix**
  1. The statement `m4 = n4--` the following.
     (a) **Copy the value of `n4` to `m4` first.**
     (b) **Decrement the value of `n2` second.**
  2. *Therefore we first copy the value 3 to* `m4`.
  3. *After the copy, we decrement* `n4` *to the value 2.*
  4. **Therefore `m4 = 3`.**
- The program output is the following.

```
prefix:  2   2                    // m3=2  n3=2
postfix: 3   2                    // m4=3  n4=2
```

4 Standalone operations

- The prefix and postfix oprators can act by themselves, without the "=" opration.
- Consider the following main program.

```
#include <iostream>

int main()
{
  int a = 1, b = 2, c = 3, d = 4;
  ++a;                                          // standalone
  b++;
  --c;
  d--;
  cout << a << "  " << b << "  " << c << "  " << d << endl;
  return 0;
}
```

- The values of a and b are incremented.
- The values of c and d are incremented.
- There is no copy, hence in this situation the prefix and postfix operators have the same effect.
- We employ standalone operations when writing increments/decrements of loop variables.

```
for (int i = 0; i < n; ++i)
for (int i = 0; i < n; i++)
for (int i = n-1; i >= 0; --i)
for (int i = n-1; i >= 0; i--)
```

5 *Why do increment and decrement operators exist?*

- The prefix increment and decrement operators are computationally efficient.
- Instead of `++x`, we could write `x = x+1`.
- Computatonally, this is what happens.
- The statement `x = x+1` means (i) store the value of x in a register in memory, (ii) store the value 1 in another register in memory, (iii) add the contents of the two registers and store the answer in the register for `x`.
- We require two memory registers and an addition operation.
- The statement `++x` means store the value of x in a register in memory and increase it by 1.
- Only one memory register is required.
- Incrementing a number by one is computationally faster than an addition operation.
    1. **The above statement may not be true for modern compilers, and may also depend on the hardware.**
    2. **The above statement also may not be valid for `float` and `double`.**
- **The prefix operation `++x` is equivalent to `x += 1`.**
- The same remarks apply for the decrement operators.
- **The prefix operation `--x` is equivalent to `x -= 1`.**