CMSC 104 - Lecture 12

John Y. Park, adapted by C Grasso

Assignment Operators

Assignment Operators

Topics

- Increment and Decrement Operators
- Assignment Operators
- Debugging Tips

Increment and Decrement Operators

- The **increment operator** ++
- The **decrement operator** --
- Precedence:
  - lower than ()
  - higher than * / and %
- Associativity: right to left
- Increment and decrement operators can only be applied to variables
  - Not to constants or expressions

Operator Precedence and Associativity

Precedence	Associativity
( )	left to right/inside-out
++ - -	right to left
! (not) - (negation)	right to left
* / %	left to right
+ (addition) - (subtraction)	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
= += -= *= /= %=	right to left

The 3 Parts of a Loop

```
#include <stdio.h>

int main () {

   int i = 1 ;          <-- init of loop control variable

   /* count from 1 to 100 */
   while ( i < 101 ) {  <-- test of loop termination

      printf ("%d ", i) ;
      i = i + 1 ;       <-- modify loop control
   }
      return 0 ;
}
```

The for Loop Repetition Structure

- The **for** loop handles details of the counter-controlled loop "automatically".
- The initialization of the the loop control variable, the termination condition test, and control variable modification are handled in the **for** loop structure.

```
for (i = 1; i < 101; i = i + 1) {
}
```

init **test** **modify**

When Does a for Loop Initialize, Test and Modify?

- Just as with a while loop, a **for** loop
  - initializes the loop control variable before beginning the first loop iteration
  - performs the loop termination test before each iteration of the loop
  - modifies the loop control variable at the very end of each iteration of the loop
- The for loop is easier to write and read for counter-controlled loops.

A for Loop That Counts From 0 to 9

```
for ( i = 0;  i < 10;  i = i + 1 )
{
  printf (“%d \n”, i) ;
}
```

We Can Count Backwards, Too

```
for ( i = 9; i >= 0; i = i - 1 )
{
  printf (“%d \n”, i) ;
}
```

We Can Count By 2's … or 7's … or Whatever

```
for ( i = 0; i < 10; i = i + 2 )
{
  printf (“%d\n”, i) ;
}
```

Increment Operator

- If we want to add one to a variable, we can say:

```
count = count + 1 ;
```

- Programs often contain statements that increment variables, so C provides these shortcuts:

```
count++ ;   OR   ++count ;
```

  Both do the same thing. They change the value of count by adding one to it.

Post-increment Operator

- The position of the **++** determines when the value is incremented. If the **++** is after the variable, then the incrementing is done last (a **post -increment**).

```
int     amount, count ;
count    =  3 ;
amount   =  2 * count++ ;
```

- After executing the last line, what values do **amount** and **count** contain?

Pre-increment Operator

- If the **++** is before the variable, then the incrementing is done first (a **pre-increment**).

```
int     amount, count ;
count    =  3 ;
amount   =  2 * ++count ;
```

- After executing the last line, what values do **amount** and **count** contain?

Code Example Using ++

```
#include <stdio.h>
int main ( )
{
     int i = 1 ;

     while ( i < 11 )
     {
         printf (“%d  ”, i) ;

         i++ ;
     }
     return 0 ;
}
```

Decrement Operator

- If we want to subtract one from a variable, we can say:

```
count = count - 1 ;
```

- Programs often contain statements that decrement variables, so C provides these shortcuts:

```
count-- ;   OR   --count ;
```

Both do the same thing. They change the value of count by subtracting one from it.

Post-decrement Operator

- If the `--` is after the variable, then the decrementing is done last (a **post-decrement**).

```
int     amount, count ;
count   =  3 ;
amount  =  2 * count-- ;
```

- After executing the last line, what values do **`amount`** and **`count`** contain?

Pre-decrement Operator

- If the `--` is before the variable, then the decrementing is done first (a **pre-decrement**).

```
int     amount, count ;
count    =  3 ;
amount   =  2 * --count ;
```

- After executing the last line, what values do **`amount`** and **`count`** contain?

A Hand Trace Example

```
int    answer, value = 4 ;
```

Code	answer	value
	garbage	4
`value = value + 1 ;`		
`value++ ;`		
`++value ;`		
`answer = 2 * value++;`		

A Hand Trace Example

```
int    answer, value = 4 ;
```

Code	value	answer
`answer = ++value / 2 ;`		
`value-- ;`		
`--value ;`		
`answer = --value * 2 ;`		
`answer = value-- / 3 ;`		

Practice

Given

```
int   a = 1,   b = 2,   c = 3 ;
```

What is the value of this expression?

```
++a * b - c--
```

What are the new values of a, b, and c?

More Practice

Given

```
int  a = 1, b = 2, c = 3, d = 4 ;
```

What is the value of this expression?

```
++b / c + a * d++
```

What are the new values of a, b, c, and d?

Assignment Operators

= += -= *= /= %=

Statement	Equivalent Statement
`a = a + 2 ;`	`a += 2 ;`
`a = a - 3 ;`	`a -= 3 ;`
`a = a * 2 ;`	`a *= 2 ;`
`a = a / 4 ;`	`a /= 4 ;`
`a = a % 2 ;`	`a %= 2 ;`
`b = b + ( c + 2 ) ;`	`b += c + 2 ;`
`d = d * ( e - 5 ) ;`	`d *= e - 5 ;`

Practice with Assignment Operators

```
int i = 1, j = 2, k = 3, m = 4 ;
```

Expression	Value
`i += j + k`	
`j *= k = m + 5`	
`k -= m /= j * 2`	

Code Example Using /= and ++
Counting the Digits in an Integer

```
#include <stdio.h>
int main ( )
{
   int num, temp, digits = 0 ;
   temp = num = 4327 ;
   while ( temp > 0  )
   {
     printf (“%d\n”, temp) ;
     temp /= 10 ;
     digits++ ;
   }
   printf (“%d digits in %d.\n”, digits, num) ;
   return 0 ;
}
```

Debugging Tips

- Trace your code by hand (a **hand trace**), keeping track of the value of each variable.

- Insert temporary **`printf`**() statements so you can see what your program is doing.
  - Confirm that the correct value(s) has been read in.
  - Check the results of arithmetic computations immediately after they are performed.