HIGH-LEVEL PROGRAMMING I

Increment/Decrement Operators

by Prasanna Ghali

Increment and Decrement Operators (1/2)

- Variables are incremented to keep track of how many times certain things have happened

```
int counter;
// some code here
counter = counter + 1;
// some more code here
```

- C/C++ provide compound assignment operator to condense incrementing

```
int counter;
// some code here
counter += 1;
// some more code here
}
```

Increment and Decrement Operators (2/2)

- To further expedite execution of these assignment statements, C/C++ provide:
  - *increment* operator **++** to increase variable's value by **1**
  - *decrement* operator **--** to decrease variable's value by **1**
- Increment and decrement operators each have two forms: *prefix* and *postfix*

Postfix-increment	`variable++`
Postfix-decrement	`variable--`

Prefix-increment	`++variable`
Prefix-decrement	`--variable`

lvalue Operands

- Operand ***expr*** for increment and decrement operators must be an *lvalue* (that is, it must be a variable)
- Result of all four expressions is *rvalue*
- *Suppose* **x** *is a variable (of any scalar type)*

Prefix Increment	Prefix Decrement	Postfix Increment	Postfix Decrement
`++10` ✗	`--x`	`x++`	`10--` ✗
`k = ++x`	`k = --x`	`k = x++`	`k = x--`
`++x = 10` ✗	`--x = 10` ✗	`x++ = 10` ✗	`x-- = 10` ✗

Postfix Increment and Decrement Operators (1/2)

- Meaning of postfix operators **`variable++`** and **`variable--`**:
  - Bump up or down **`variable`** ***after*** using its original value in surrounding expression
- Suppose we've variable **`x`** of type **`int`**

Contents of x before	Expression	Value of expression	Contents of x after sequence point
10	x++	10	11
10	x--	10	9

Postfix Increment and Decrement Operators (2/2)

Contents of x before	Expression	Value of expression	Contents of x after sequence point
10	x++	10	11
10	x--	10	9

- If `i` and `j` are `int` variables, statement `i=j--;` can be equivalently rewritten in any of 3 ways:

```
i = j;
j = j - 1;
```

```
i = j;
j--;
```

```
i = j;
--j;
```

Compiler's Perspective: Postfix Operators

- From compiler's perspective, expression **`variable++`** means
  - Get location of **`variable`** in memory
  - Load contents at this location into CPU register, say **`r0`**
  - Use value in **`r0`** register in expression
  - Increment value in register **`r0`**
  - Store incremented value in **`r0`** to location of **`variable`** in memory
  - Side effect of storing incremented value in **`variable`** will occur at next sequence point

Prefix Increment and Decrement Operators (1/2)

- Meaning of prefix operators **`++variable`** and **`--variable`**:
  - Bump up or down **`variable`** ***before*** using its value in surrounding expression
- Suppose we've variable **`x`** of type **`int`**

Contents of x before	Expression	Contents of x after	Value of expression
10	++x	11	11
10	--x	9	9

Prefix Increment and Decrement Operators (2/2)

Contents of x before	Expression	Contents of x after	Value of expression
10	++x	11	11
10	--x	9	9

- If `i` and `j` are `int` variables, statement `i=--j;` can be equivalently rewritten in any of 3 ways:

```
j = j - 1;
i = j;
```

```
j--;
i = j;
```

```
--j;
i = j;
```

Prefix Increment and Decrement Operators (2/2)

- From compiler's perspective, expression **`++variable`** means
  - Get location of **`variable`** in memory
  - Load contents at this location into a CPU register
  - Increment contents of this CPU register
  - Store incremented value in register to location of **`variable`** in memory
  - Use incremented value in register in expression

Equivalence

- Following statements have same effect:

Assignment	Compound Assignment	Prefix Increment	Postfix Increment
`x = x + 1;`	`x += 1;`	`++x;`	`x++;`

Assignment	Compound Assignment	Prefix Decrement	Postfix Decrement
`x = x - 1;`	`x -= 1;`	`--x;`	`x--;`

Precedence and Associativity

high to low precedence order

Operator	Meaning	Associativity
( )	parentheses or grouping	L-R
++ --	Postfix increment/decrement	L-R
++ --	Prefix increment/decrement	R-L
+ -	unary plus, unary minus	R-L
* / %	multiplication, division, remainder	L-R
+ -	addition, subtraction	L-R
< <= > >=	relational	L-R
== !=	Equivalence	L-R
&&	logical AND	L-R
||	logical OR	L-R
=	assignment	R-L
,	comma	L-R

Increment and Decrement Operators: Example 0 (1/2)

- Assume all variables are of type `int` with `a` initialized to value `5` before each statement
- Provide values ***after*** statement is executed …

Statement	Value of a	Value of b	Value of c
`c = a++;`		-	
`c = ++a;`		-	
`c = b = a++;`			
`c = b = ++a;`			

Increment and Decrement Operators: Example 0 (2/2)

- Assume all variables are of type `int` with `a` initialized to value `5` before each statement
- Provide values ***after*** statement is executed …

Statement	Value of a	Value of b	Value of c
`c = a++;`	6	-	5
`c = ++a;`	6	-	6
`c = b = a++;`	6	5	5
`c = b = ++a;`	6	6	6

Increment and Decrement Operators: Example 1 (1/2)

- Assume all variables are of type `int` with `a` and `b` initialized to values `5` and `3` before each statement
- Provide values after statement is executed …

Statement	Value of a	Value of b	Value of c
`c = a++ + b++;`			
`c = ++a + b++;`			
`c = a++ + ++b;`			
`c = ++a + ++b;`			

Increment and Decrement Operators: Example 1 (2/2)

- Assume all variables are of type `int` with `a` and `b` initialized to values `5` and `3` before each statement
- Provide values after statement is executed …

Statement	Value of a	Value of b	Value of c
`c = a++ + b++;`	6	4	8
`c = ++a + b++;`	6	4	9
`c = a++ + ++b;`	6	4	9
`c = ++a + ++b;`	6	4	10

Increment and Decrement Operators: Example 2 (1/2)

- Write comma separated list of tokens extracted by compiler (indicate whether expression is legal or not)
  - Recall from earlier part of course that a ***token*** is *group of characters that cannot be split up without changing their meaning*

Expression	List of Tokens	Legal?
j+++k		
j++++k		
j+++++k		
j+++ ++k		

Increment and Decrement Operators: Example 2 (2/2)

- Write comma separated list of tokens extracted by compiler (indicate whether expression is legal or not)
  - Recall from earliest part of course that a ***token*** is *group of characters that cannot be split up without changing their meaning*

Expression	List of Tokens	Legal
j+++k	j, ++, +, k	√
j++++k	j, ++, ++, k	×
j+++++k	j, ++, ++, +, k	×
j+++ ++k	j, ++, +, ++, k	√

Increment and Decrement Operators: Example 3 (1/2)

- Which of following expressions are legal?
- Assume variables `j` and `k` are of type `int` and are initialized with values `1` and `2`

Expression	Legal or not?	If legal, value of expression
`j+++k`		
`j++++k`		
`j+++++k`		
`j+++ ++k`		

Increment and Decrement Operators: Example 3 (2/2)

- Which of following expressions are legal?
- Assume variables `j` and `k` are of type `int` and are initialized with values `1` and `2`

Expression	Legal or not?	If legal, value of expression
`j+++k`	√	3
`j++++k`	×	-
`j+++++k`	×	-
`j+++ ++k`	√	4

Increment and Decrement Operators: Example 4 (1/2)

- Are following expressions evaluated unambiguously?
- Assume variable `j` is of type `int` and is initialized with value `1`

Expression	Unambiguous?	If unambiguous, value of expression
`j++*j++`		
`++j*++j`		
`++j*j++`		
`j++*++j`		

Increment and Decrement Operators: Example 4 (2/2)

- Are following expressions evaluated unambiguously?
- Assume variable `j` is of type `int` and is initialized with value **1**

Expression	Unambiguous?	If unambiguous, value of expression
`j++*j++`	No	Unspecified behavior
`++j*++j`	No	Unspecified behavior
`++j*j++`	No	Unspecified behavior
`j++*++j`	No	Unspecified behavior

Increment and Decrement Operators: Example 5

- What is printed to standard output by code fragment?

```c
#include <stdio.h>
#include <stdbool.h>

int main(void) {
  int i = 1, j = 1, k = 1;
  bool flag = ++i || ++j && ++k;
  if (flag == true) {
    printf("flag is true\n");
  }
  printf("%d %d %d\n", i, j, k);

  i = 7; j = 8; k = 9;
  flag = i - 7 && j++ < k;
  if (flag == false) {
    printf("flag is false\n");
  }
  printf("%d %d %d\n", i, j, k);
}
```