

Programming Languages

Dr. H. Irem Türkmen

Contact

- Contact info
 - office : D-033
 - e-mail irem@yildiz.edu.tr
- When to contact
 - e-mail first,
 - take an appointment

Course Outline

- **Review of “Introduction to Computer Engineering II” course**
 - data types, control flow, and so on...
- **Arrays**
 - Strings, multi-dimensional arrays, pointers
- **Dynamic memory allocation**
 - multi-dimensional
- **Functions**
 - parameter passing, return values
 - recursion
 - function pointers

Course Outline Cont'd

- **Structures**
 - structures, unions
 - linked lists
- **Storage classes**
 - scope & duration
- **C preprocessors**
- **FILE I/O**

Course Material

- **Lecture notes**
 - slides available at www.ce.yildiz.edu.tr
- **Web resources**
 - Always make reference to resources !
- **Book** : Darnell P. A. and Margolis P. E., C: A Software Engineering Approach, 1996 (3rd) edition


Precedence & associativity

- All operators have two important properties called ***precedence*** and ***associativity***.
 - Both properties affect how operands are attached to operators
- Operators with higher precedence have their operands bound, or grouped, to them before operators of lower precedence, regardless of the order in which they appear.
- In cases where operators have the same precedence, associativity (sometimes called binding) is used to determine the order in which operands grouped with operators.


- $2 + 3 * 4$

- $a = b = c;$

Precedence & associativity

Class of operator	Operators in that class	Associativity	Precedence
primary	() [] -> .	Left-to-Right	 HIGHEST
unary	cast operator sizeof & (address of) * (dereference) - + ~ ++ -- !	Right-to-Left	
multiplicative	* / %	Left-to-Right	
additive	+ -	Left-to-Right	
shift	<< >>	Left-to-Right	
relational	< <= > >=	Left-to-Right	
equality	== !=	Left-to-Right	

Precedence & associativity

Class of operator	Operators in that class	Associativity	Precedence
bitwise AND	&	Left-to-Right	 LOWEST
bitwise exclusive OR	^	Left-to-Right	
bitwise inclusive OR	 	Left-to-Right	
logical AND	&&	Left-to-Right	
logical OR	 	Left-to-Right	
conditional	? :	Right-to-Left	
assignment	= += -= *= /= %= >>= <<= &= ^=	Right-to-Left	
comma	,	Left-to-Right	

Parenthesis

- The compiler groups operands and operators that appear within the parentheses first, so you can use parentheses to specify a particular grouping order.
 - $(2 - 3) * 4$
 - $2 - (3 * 4)$

- The inner most parentheses are evaluated first. The expression $(3+1)$ and $(8-4)$ are at the same depth, so they can be evaluated in either order.

$$1 + ((3+1) / (8 - 4) - 5)$$

$$1 + (4 / (8 - 4) - 5)$$

$$1 + (4 / 4 - 5)$$

$$1 + (1 - 5)$$

$$1 + -4$$

$$-3$$

binary arithmetic operators

Operator	Symbol	Form	Operation
multiplication	*	$\mathbf{x * y}$	x times y
division	/	$\mathbf{x / y}$	x divided by y
remainder	%	$\mathbf{x \% y}$	remainder of x divided by y
addition	+	$\mathbf{x + y}$	x plus y
subtraction	-	$\mathbf{x - y}$	x minus y

The remainder operator

- Unlike other arithmetic operators, which accept both integer and floating point operands, the remainder operator accepts only integer operands!
- If either operand is negative, the remainder can be negative or positive, depending on the implementation

arithmetic assignment operators

Operator	Symbol	Form	Operation
assign	=	$a = b$	put the value of b into a
add-assign	+=	$a += b$	put the value of $a+b$ into a
subtract-assign	-=	$a -= b$	put the value of $a-b$ into a
multiply-assign	*=	$a *= b$	put the value of $a*b$ into a
divide-assign	/=	$a /= b$	put the value of a/b into a
remainder-assign	%=	$a \% = b$	put the value of $a\%b$ into a

arithmetic assignment operators

```
int m = 3, n = 4;
```

```
float x = 2.5, y = 1.0;
```

```
m += n + x - y
```

```
m = (m + ((n+x) -y ))      (8)
```

```
m /= x * n + y
```

```
m = (m / ((x*n) + y ))    (0)
```

```
n %= y + m
```

```
n = (n % (y + m) )        (invalid operants)
```

```
x += y -= m
```

```
x = ( x + ( y = (y - m ))) (0.5)
```

increment & decrement operators

Operator	Symbol	Form	Operation
postfix increment	++	a++	get value of a, then increment a
postfix decrement	--	a--	get value of a, then decrement a
prefix increment	++	++a	increment a, then get value of a
prefix decrement	--	--b	decrement a, then get value of a

increment & decrement operators

```
main () {  
    int j=5, k=5;  
    printf("j: %d\t k : %d\n", j++, k--);  
    printf("j: %d\t k : %d\n", j, k);  
    return 0;  
}
```

```
j:5 k:5  
j:6 k:4
```

```
main () {  
    int j=5, k=5;  
    printf("j: %d\t k : %d\n", ++j, --k);  
    printf("j: %d\t k : %d\n", j, k);  
    return 0;  
}
```

```
j: 6 k:4  
j: 6 k:4
```

increment & decrement operators

```
int j = 0, m = 1, n = -1, i = 5;
```

```
m++ --j
```

```
m += ++j * 2
```

```
x = i * i++
```

```
(m++) - (--j)           (1 - -1 = 2)
```

```
m = ( m + ((++j) * 2 )   (1 + 1 * 2 = 3)
```

```
x = j * (j++)
```

(implementation dependent 25 or 30)

comma operator

- Comma operator allows you to evaluate two or more distinct expressions wherever a single expression allowed!
 - `for (j = 0, k = 100; k - j > 0; j++, k--)`

relational operators

Operator	Symbol	Form	Result
greater than	>	a > b	1 if a is greater than b; else 0
less than	<	a < b	1 if a is less than b; else 0
greater than or equal to	>=	a >= b	1 if a is greater than or equal to b; else 0
less than or equal to	<=	a <= b	1 if a is less than or equal to b; else 0
equal to	==	a == b	1 if a is equal to b; else 0
not equal to	!=	a != b	1 if a is NOT equal to b; else 0

relational operators

```
int j=0, m=1, n=-1;  
float x=2.5, y=0.0;
```

$j > m$

$m/n < x$

$j \leq m \geq n$

$++j == m != y * 2$

$j > m$ (0)

$(m / n) < x$ (1)

$((j \leq m) \geq n)$ (1)

$((++j) == m) != (y * 2)$ (1)

logical operators

Operator	Symbol	Form	Result
logical AND	&&	a && b	1 if a and b are non zero; else 0
logical OR	 	a b	1 if a or b is non zero; else 0
logical negation	!	!a	1 if a is zero; else 0

bit manipulation operators

Operator	Symbol	Form	Result
right shift	>>	$x \gg y$	x shifted right by y bits
left shift	<<	$x \ll y$	x shifted left by y bits
bitwise AND	&	$x \& y$	x bitwise ANDed with y
bitwise inclusive OR		$x y$	x bitwise ORed with y
bitwise exclusive OR (XOR)	^	$x \wedge y$	x bitwise XORed with y
bitwise complement	~	$\sim x$	bitwise complement of x