

Special Operators

Lecture 9

1

Conditional Operator

- The conditional operator ?: takes three operands
 - $c ? r1 : r2$
 - The value of the expression using the conditional operator is the value of either its second or third operand, depending on the value of the first operand
 - Same as

```
if c
  result value is r1
else
  result value is r2
```

2

Conditional Operator

➤ Examples

- In assignment

```
x = (a < b) ? a : b;
```

→ x will be assigned the smallest value between a and b

- In assignment

```
y = (a == b) ? (a + b) : (a - b);
```

→ if (a == b), y will be assigned (a + b)

→ if (a != b), y will be assigned (a - b)

3

Conditional Operator

- Very useful for macros

■ Examples

```
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
```

→ Returns the max between the parameters assigned to a and b.

```
#define ISLETTER(c) (((c) >= 'A' && (c) <= 'z') ? 1 : 0)
```

→ Returns 1 if the value assigned to c is a letter and returns 0 if not.

4

Sequential Evaluation

- **The comma operator**
 - Evaluates its two operands in sequence, yielding the value of the second operand as the value of the expression
 - The value of the first is discarded

5

Sequential Evaluation

- **Example**
 - In assignments
`x = (i += 2, a[i]);` → `i += 2; x = a[i];`
→ Parentheses are important because precedence of the assignment operator is higher than precedence of the comma
 - In for loops
`for (i = 0, j = 0; i < I_MAX && j < J_MAX; i += 2, j += i)`
`printf ("i = %d, j = %d\n", i, j);`

6

Bitwise Operators

- **Positive integers are represented in the computer by standard binary numbers**
 - Examples:
`short n = 13;`
→ in memory - 0000 0000 0000 1101
→ $2^0 + 2^2 + 2^3 = 13$

`char c = 5;`
→ in memory - 0000 0101
→ $2^0 + 2^2 = 5$

7

Bitwise Operators

- **Bitwise operators**
 - take operands of any integer type
 - `char, short, int, long`
 - but treat an operand as a collection of bits rather than a single number

8

Bitwise Negation

■ Bitwise negation

➤ Operand ~

- Application of ~ to an integer produces a value in which each bit of the operand has been replaced by its negation

- 0 becomes 1
- 1 becomes 0

• Example

n = 0000 0000 0000 1101

~n = 1111 1111 1111 0010

9

Bitwise Shift

■ Shift operators

➤ shift left → <<

➤ shift right → >>

■ Take two integers operands

➤ The value on the left is the number to be shifted

- Viewed as a collection of bits that can move
- To avoid implementation problems, avoid negative numbers when shifting right

➤ The value on the right is a nonnegative number telling how far to move the bits

10

Bitwise Shift

■ Operand

➤ << shifts bits left

➤ >> shifts bits right

■ The bits that “fall off the end” are lost

■ The “emptied” positions are filled with zeros

11

Bitwise Shift

■ Example:

n = 0000 0000 0000 1101

n << 1 → 0000 0000 0001 1010

(lost 1 bit on the left)

n << 4 → 0000 0000 1101 0000

(lost 4 bits on the left)

n >> 3 → 0000 0000 0000 0001

(lost 3 bits on the right)

12

Bitwise Shift

- Compound assignment operators `<<=` and `>>=`
 - cause the value resulting from the shift to be stored in the variable supplied as the left operand

13

Bitwise AND, XOR, and OR

- The bitwise operators `&` (and), `^` (xor), and `|` (or)
 - Take two operands that are viewed as strings of bits
 - The operator determines each bit of the result by considering corresponding bits of each operand
 - For each bit i
 - $r_i = n_i \& m_i \rightarrow 1$ when both n_i and m_i are 1
 - $r_i = n_i | m_i \rightarrow 1$ when n_i and/or m_i is 1
 - $r_i = n_i ^ m_i \rightarrow 1$ when n_i and m_i do not match

14

Bitwise AND, XOR, and OR

■ Example:

```
n = 0000 0000 0000 1101
m = 0000 0000 0011 1100
m & n = 0000 0000 0000 1100
```

```
n = 0000 0000 0000 1101
m = 0000 0000 0011 1100
m | n = 0000 0000 0011 1101
```

15

Bitwise AND, XOR, and OR

■ Example:

```
n = 0000 0000 0000 1101
m = 0000 0000 0011 1100
m ^ n = 0000 0000 0011 0001
```

16

Bitwise AND, XOR, and OR

- **Compound assignment operators $&=$, $|=$, and $\wedge=$**
 - cause the resulting value to be stored in the variable supplied as the left operand

17

Bitwise Operators

- **Notes on shifting**
 - \ll by 1 is the same as multiplying by 2
 - \gg by 1 is the same as dividing by 2
- **Notes on \sim and $!$**
 - \sim and $!$ are different operators
 - \sim is a bitwise operator
 - each bit is reversed
 - $!$ is a logical complement or negation
 - !nonzero \rightarrow false (zero)
 - !zero \rightarrow true (one)

18

Bitwise Operators

- **Notes on AND**
 - $x \& 0$ is always 0
 - $x \& 1$ is always x
- **Notes on OR**
 - $x | 1$ is always 1
 - $x | 0$ is always x

19

Bitwise Operators

- **Masks -- Used to change specific bits in an integer**
 - **To set specific bits**
 - Use OR with a mask in which only the bits to be set have 1

```
short c =      0000 0101;
short mask =   0000 0010;
c | mask ==    0000 0111
```

 - **To zero specific bits**
 - Use AND with a mask in which only the bits to be zeroed have 0

```
short c =      0000 0101;
short mask =   1111 1110;
c & mask ==    0000 0100
```

20

Bitwise Operators

■ Masks -- Used to change specific bits in an integer

➤ To verify specific bit

- Use AND with a mask in which only the bit to be verified is 1
- Result == 0 implies that bit == 0
- Result != 0 implies that bit == 1

```
short c =      0000 0101;  
short mask =   0000 0100;  
c & mask ==   0000 0100 (not zero ==> bit is not zero)
```

21

Bitwise Operators

■ Notes on XOR

- $x \wedge 0$ is always x
- $x \wedge 1$ is always $\sim x$
- $x \wedge x$ is always 0
- $x \wedge \sim x$ is always 1
- if $x \wedge y == z$, then
 - $x == z \wedge y$ and $y == z \wedge x$

22