	Operators	
	• Expressions are built from variables, constants, and operators.	
Chapter 4	• C has a rich collection of operators, including	
Expressions	- arithmetic operators	
	– relational operators	
	– logical operators	
	<ul> <li>assignment operators</li> <li>increment and decrement operators</li> </ul>	
	- increment and decrement operators	
CPROGRAMMING 1 Copyright © 2008 W. W. Norton & Company. All rights reserved. All rights reserved.	CPROGRAMMING A Moderni Approach iscess terries 2 Copyright © 2008 W. W. Norton & Comp All rights reserved.	
	A MOUTR Approuch sicessiantes	
	л этошет п прргоцон засеке венек	
Chapter 4: Expressions	Chapter 4: Expressions	
Chapter 4: Expressions Arithmetic Operators	Chapter 4: Expressions	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction	Chapter 4: Expressions Binary Arithmetic Operators • The value of i % j is the remainder when i is	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction * multiplication / division	Chapter 4: Expressions Binary Arithmetic Operators • The value of i % j is the remainder when i is divided by j. 10 % 3 has the value 1, and 12 % 4 has the value 0. • Binary arithmetic operators—with the exception	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction * multiplication / division % remainder	Chapter 4: Expressions         Binary Arithmetic Operators         • The value of i % j is the remainder when i is divided by j.         10 % 3 has the value 1, and 12 % 4 has the value 0.         • Binary arithmetic operators—with the exception of %—allow either integer or floating-point	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction * multiplication / division	<ul> <li>Chapter 4: Expressions</li> <li>Binary Arithmetic Operators</li> <li>The value of i % j is the remainder when i is divided by j. <ol> <li>10 % 3 has the value 1, and 12 % 4 has the value 0.</li> </ol> </li> <li>Binary arithmetic operators—with the exception of %—allow either integer or floating-point operands, with mixing allowed.</li> </ul>	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction * multiplication / division % remainder	Chapter 4: Expressions         Binary Arithmetic Operators         • The value of i % j is the remainder when i is divided by j.         10 % 3 has the value 1, and 12 % 4 has the value 0.         • Binary arithmetic operators—with the exception of %—allow either integer or floating-point	
Chapter 4: Expressions Arithmetic Operators C provides five binary arithmetic operators: + addition - subtraction * multiplication / division % remainder	<ul> <li>Chapter 4: Expressions</li> <li>Binary Arithmetic Operators</li> <li>The value of i % j is the remainder when i is divided by j. <ol> <li>10 % 3 has the value 1, and 12 % 4 has the value 0.</li> </ol> </li> <li>Binary arithmetic operators—with the exception of %—allow either integer or floating-point operands, with mixing allowed.</li> <li>When int and float operands are mixed, the</li> </ul>	

#### Chapter 4: Expressions

# The / and % Operators

- The / and % operators require special care:
  - When both operands are integers, / "truncates" the result. The value of 1 / 2 is 0, not 0.5.
  - The % operator requires integer operands; if either operand is not an integer, the program won't compile.
  - Using zero as the right operand of either / or % causes undefined behavior.

5

 In C99, the result of a division is always truncated toward zero and the value of i % j has the same sign as i.

#### Chapter 4: Expressions

# **Operator Precedence**

• The arithmetic operators have the following relative precedence:

Highest: + - (unary) \* / % Lowest: + - (binary)

• Examples:

i + j \* k is equivalent to i + (j \* k)-i \* -j is equivalent to (-i) \* (-j)+i + j / k is equivalent to (+i) + (j / k)

6



**C**PROGRAMMING

### Chapter 4: Expressions Chapter 4: Expressions **Assignment Operators** Simple Assignment • The effect of the assignment v = e is to evaluate • Simple assignment: used for storing a value into a variable the expression *e* and copy its value into *v*. • Compound assignment: used for updating a value • *e* can be a constant, a variable, or a more already stored in a variable complicated expression: i = 5; /\* i is now 5 \*/ /\* j is now 5 \*/ j = i; k = 10 \* i + j; /\* k is now 55 \*/ Copyright © 2008 W. W. Norton & Company. All rights reserved. Copyright © 2008 W. W. Norton & Company. All rights reserved. **C**PROGRAMMING **C**PROGRAMMING 7 8 Chapter 4: Expressions

# **Compound Assignment**

- Assignments that use the old value of a variable to compute its new value are common.
- Example:
- i = i + 2;
- Using the += compound assignment operator, we simply write:

```
i += 2;
          /* same as i = i + 2; */
```

• There are other compound assignment operators, including the following:

9

-= \*= /= %=

**C**PROGRAMMING

#### Chapter 4: Expressions

## Increment and Decrement Operators

- Two of the most common operations on a variable are "incrementing" (adding 1) and "decrementing" (subtracting 1):
  - i = i + 1;

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

• Incrementing and decrementing can be done using the compound assignment operators:

10

i += 1; j −= 1;

**C**PROGRAMMING

Copyright © 2008 W. W. Norton & Company All rights reserved.

## Chapter 4: Expressions

## Increment and Decrement Operators

- C provides special ++ (*increment*) and --(*decrement*) operators.
- The ++ operator adds 1 to its operand. The -operator subtracts 1.
- The increment and decrement operators are tricky to use:
  - They can be used as *prefix* operators (++i and --i) or *postfix* operators (i++ and i--).
  - They have side effects: they modify the values of their operands.

11

```
CPROGRAMMING
```

Copyright © 2008 W. W. Norton & Company. All rights reserved.

Copyright © 2008 W. W. Norton & Company. All rights reserved.

#### Chapter 4: Expressions

## Increment and Decrement Operators

• Evaluating the expression ++i (a "pre-increment") yields i + 1 and—as a side effect—increments i:

```
i = 1;
printf("i is %d\n", ++i); /* prints "i is 2" */
printf("i is %d\n", i);
                          /* prints "i is 2" */
```

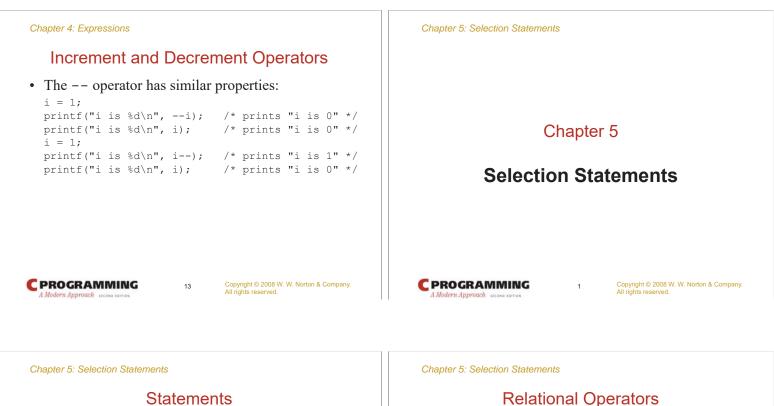
• Evaluating the expression i++ (a "post-increment") produces the result i, but causes i to be incremented afterwards:

```
i = 1:
printf("i is %d\n", i++); /* prints "i is 1" */
                          /* prints "i is 2" */
printf("i is %d\n", i);
```

12

**C**PROGRAMMING

```
Copyright © 2008 W. W. Norton & Company.
All rights reserved.
```



- ....
- Most of C's statements fall into three categories:
   *Selection statements:* if and switch
  - Iteration statements: while, do, and for
  - Jump statements: break and continue
  - (return also falls in this category.)
- Other C statements:
  - Compound statement
  - Null statement

**C**PROGRAMMING

Chapter 5: Selection Statements

# **Equality Operators**

2

- C provides two equality operators:
  - == equal to
  - ! = not equal to
- The equality operators are left associative and produce either 0 (false) or 1 (true) as their result.
- The equality operators have lower precedence than the relational operators, so the expression

4

```
i < j == j < k
```

```
is equivalent to
```

```
(i < j) == (j < k)
```

**C**PROGRAMMING

Copyright © 2008 W. W. Norton & Company. All rights reserved.

Copyright © 2008 W. W. Norton & Company. All rights reserved.

Chapter 5: Selection Statements

**C**PROGRAMMING

• C's relational operators:

<= less than or equal to

used in expressions.

>= greater than or equal to

less than greater than

<

>

# **Logical Operators**

• These operators produce 0 (false) or 1 (true) when

• The relational operators can be used to compare

3

integers and floating-point numbers, with

operands of mixed types allowed.

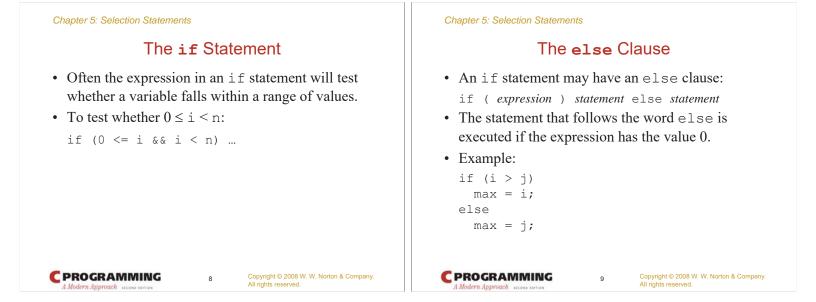
- More complicated logical expressions can be built from simpler ones by using the *logical operators:* 
  - ! logical negation
  - & & logical and
  - || logical or
- The ! operator is unary, while & & and || are binary.
- The logical operators produce 0 or 1 as their result.
- The logical operators treat any nonzero operand as a true value and any zero operand as a false value.

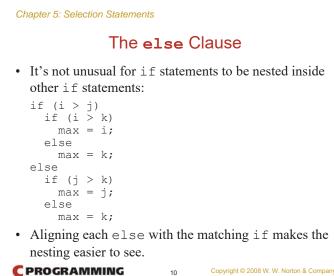
5

**C**PROGRAMMING

Copyright © 2008 W. W. Norton & Company. All rights reserved.

Chapter 5: Selection Statements	Chapter 5: Selection Statements
The if Statement	The if Statement
<ul> <li>The if statement allows a program to choose between two alternatives by testing an expression.</li> <li>In its simplest form, the if statement has the form if ( <i>expression</i> ) <i>statement</i></li> <li>When an if statement is executed, <i>expression</i> is evaluated; if its value is nonzero, <i>statement</i> is executed.</li> <li>Example:     if (line_num == MAX_LINES)         line_num = 0;</li> </ul>	<ul> <li>Confusing == (equality) with = (assignment) is perhaps the most common C programming error.</li> <li>The statement <pre>if (i == 0) tests whether i is equal to 0.</pre> </li> <li>The statement <pre>if (i = 0 &lt; j) assigns 0 to i, then tests whether the result is nonzero.</pre></li></ul>
CPROGRAMMING A Modern Approach access tarres 6 Copyright © 2008 W. W. Norton & Company. All rights reserved.	CPROGRAMMING 7 Copyright © 2008 W. W. Norton & Company. A Modern Approach LICENS ISSNESS





**C**PROGRAMMING

Chapter 5: Selection Statements

# Cascaded if Statements

• A "cascaded" if statement is often the best way to test a series of conditions, stopping as soon as one of them is true.

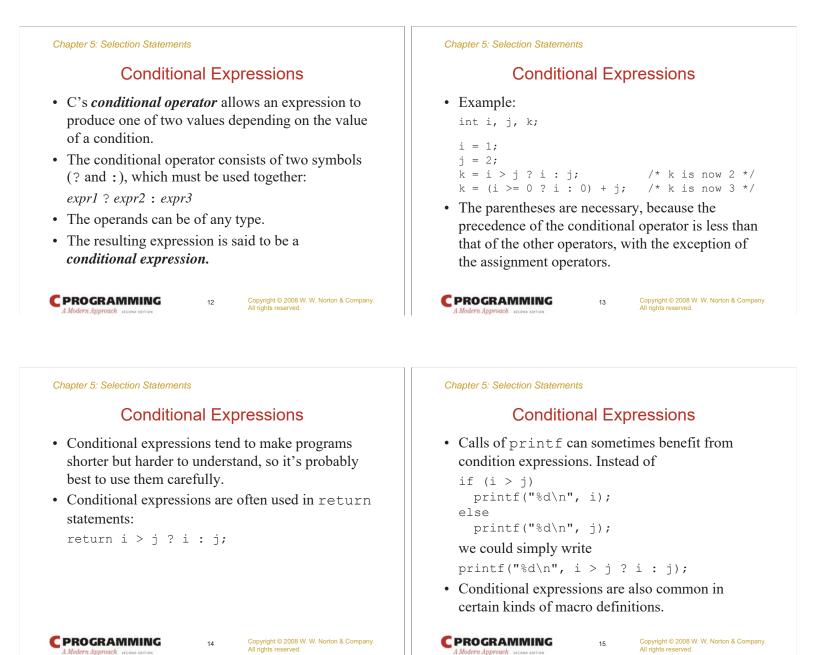
```
• Example:
 if (n < 0)
```

**C**PROGRAMMING

```
printf("n is less than 0 \in ;
else
  if (n == 0)
    printf("n is equal to 0\n");
  else
    printf("n is greater than 0 \in ;
```

11

Copyright © 2008 W. W. Norton & Company. All rights reserved.



Chapter 5: Selection Statements The switch Statement • A cascaded if statement can be used to compare an expression against a series of values: if (qrade == 4)printf("Excellent"); else if (grade == 3) printf("Good"); else if (grade == 2) printf("Average"); else if (grade == 1) printf("Poor"); else if (grade == 0) printf("Failing"); else printf("Illegal grade"); **C**PROGRAMMING Copyright © 2008 W. W. Norton & Company. All rights reserved. 16

Chapter 5: Selection Statements

}

## The switch Statement

• The switch statement is an alternative:

```
switch (grade) {
    case 4: printf("Excellent");
               break;
    case 3: printf("Good");
              break;
    case 2: printf("Average");
               break;
    case 1: printf("Poor");
               break;
    case 0: printf("Failing");
              break;
    default: printf("Illegal grade");
               break;
CPROGRAMMING
                                Copyright © 2008 W. W. Norton & Company.
All rights reserved.
                         17
```

#### Chapter 5: Selection Statements Chapter 5: Selection Statements The switch Statement The switch Statement • A switch statement may be easier to read than a • The word switch must be followed by an integer cascaded if statement. expression-the controlling expression-in • switch statements are often faster than if parentheses. • Characters are treated as integers in C and thus can statements. be tested in switch statements. • Most common form of the switch statement: • Floating-point numbers and strings don't qualify, switch ( expression ) { case constant-expression : statements however. case constant-expression : statements default : *statements* } Copyright © 2008 W. W. Norton & Company All rights reserved. Copyright © 2008 W. W. Norton & Company. All rights reserved. **C**PROGRAMMING **C**PROGRAMMING 18 19 Chapter 5: Selection Statements Chapter 5: Selection Statements

## The switch Statement

- Each case begins with a label of the form case constant-expression :
- A *constant expression* is much like an ordinary expression except that it can't contain variables or function calls.
  - 5 is a constant expression, and 5 + 10 is a constant expression, but n + 10 isn't a constant expression (unless n is a macro that represents a constant).
- The constant expression in a case label must evaluate to an integer (characters are acceptable).

**C**PROGRAMMING Copyright © 2008 W. W. Norton & Company All rights reserved. 20

## The switch Statement

- After each case label comes any number of statements.
- No braces are required around the statements.
- The last statement in each group is normally break.

**C**PROGRAMMING

Copyright © 2008 W. W. Norton & Company All rights reserved.

#### Chapter 5: Selection Statements

**C**PROGRAMMING

# The switch Statement

- Duplicate case labels aren't allowed.
- The order of the cases doesn't matter, and the default case doesn't need to come last.
- Several case labels may precede a group of statements:

22

Copyright © 2008 W. W. Norton & Company. All rights reserved.

```
switch (grade) {
 case 4:
 case 3:
  case 2:
  case 1: printf("Passing");
           break;
  case 0: printf("Failing");
           break;
  default: printf("Illegal grade");
           break;
}
```

Chapter 5: Selection Statements

# Program: Printing a Date in Legal Form

21

• Contracts and other legal documents are often dated in the following way:

Dated this day of , 20 .

- The date.c program will display a date in this form after the user enters the date in month/day/year form: Enter date (mm/dd/yy): 7/19/14 Dated this 19th day of July, 2014.
- The program uses switch statements to add "th" (or "st" or "nd" or "rd") to the day, and to print the month as a word instead of a number.

23

**C**PROGRAMMING

Chapter 5: Selection Statements	S			
date.c				
/* Prints a date in legal f	form */			
#include <stdio.h></stdio.h>				
<pre>int main(void) {     int month, day, year;</pre>				
printf("Enter date (mm/do scanf("%d /%d /%d", &mont		&year);		
<pre>printf("Dated this %d", c switch (day) { case 1: case 21: case 3 printf("st"); break; case 2: case 22: printf("nd"); break; case 3: case 23: printf("rd"); break; default: printf("th"); } printf(" day of ");</pre>	31:			
C PROGRAMMING A Modern Approach SECOND RETING	24	Copyright © 2008 W. W. Norton & Company. All rights reserved.		

Chapter 5: Selection Statements					
switch (mo	nth) {				
case 1:	<pre>printf("January");</pre>	break;			
case 2:	<pre>printf("February");</pre>	break;			
case 3:	printf("March");	break;			
case 4:	<pre>printf("April");</pre>	break;			
case 5:	<pre>printf("May");</pre>	break;			
case 6:	<pre>printf("June");</pre>	break;			
case 7:	<pre>printf("July");</pre>	break;			
case 8:	<pre>printf("August");</pre>	break;			
case 9:	<pre>printf("September");</pre>	break;			
case 10:	<pre>printf("October");</pre>	break;			
case 11:	<pre>printf("November");</pre>	break;			
case 12:	<pre>printf("December");</pre>	break;			
}					
printf(", 20%.2d.\n", year);					
return 0;					
}					

CPROGRAMMING 25

Copyright © 2008 W. W. Norton & Company. All rights reserved.