Chapter 4 part 1
C Program Control

C How to Program

© By Pearson Education, Inc. All Rights Reserved.

for Repetition Statement

```
for ( counter = 1; counter <= 10; counter++ ) {
   printf( "%u\n", counter );
}
```

- When the `for` statement begins executing, the control variable `counter` is initialized to `1`.
- Then, the loop-continuation condition `counter <= 10` is checked.
- `%u` is used for unsigned int

© By Pearson Education, Inc. All Rights Reserved.

for Repetition Statement (Cont.)

```
for ( counter = 1; counter <= 10; counter++ ) {
  printf( "%u\n", counter );
}
```

- Because the initial value of `counter` is `1`, the condition is satisfied, so the `printf` statement prints the value of `counter`, namely `1`.
- The control variable `counter` is then incremented by the expression `counter++`, and the loop begins again with the loop-continuation test.

© By Pearson Education, Inc. All Rights Reserved.

for Repetition Statement (Cont.)

```
for ( counter = 1; counter <= 10; counter++ ) {
   printf( "%u\n", counter );
}
```

- Because the control variable is now equal to 2, the final value is not exceeded, so the program performs the printf statement again.
- This process continues until the control variable counter is incremented to its final value of 11—this causes the loop-continuation test to fail, and repetition terminates.

© By Pearson Education, Inc. All Rights Reserved.

Type in Codeblocks

main.c

```c
// Counter controlled repetition
#include <stdio.h>

int main()
{
    unsigned int counter;

    for ( counter = 1; counter <= 10; counter++ ) {
        printf( "%u\n", counter );
    }

    return 0;
}
```

© By Pearson Education, Inc. All Rights Reserved.

for Statement Header Components

- Notice that the `for` statement "does it all"—it specifies each of the items needed for counter-controlled repetition with a control variable.

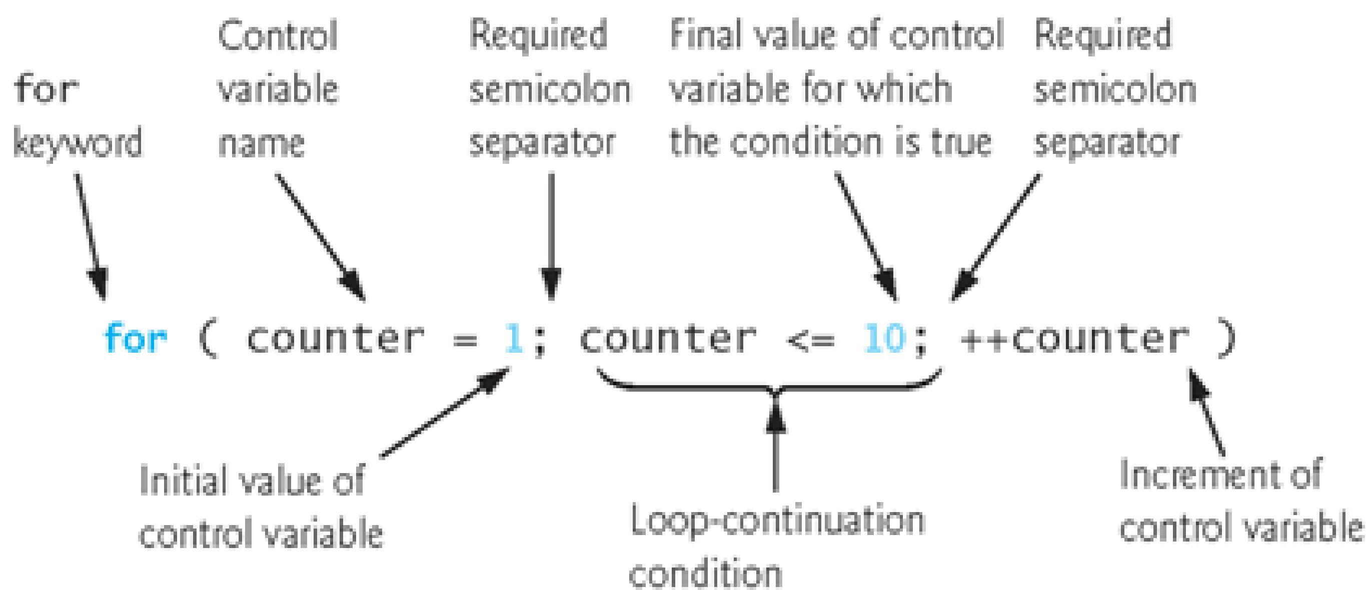


Fig. 4.3 | for statement header components.

Increment Expression Acts Like a Standalone Statement

- The increment expression in the `for` statement acts like a stand-alone C statement at the end of the body of the `for`.
- Therefore, the expressions

```
counter = counter + 1
counter += 1
++counter
counter++
```

  are all equivalent in the increment part of the `for` statement.
- Because the variable being preincremented or postincremented here does not appear in a larger expression, both forms of incrementing have the same effect.
- The two semicolons in the `for` statement are required.

© By Pearson Education, Inc. All Rights Reserved.

Off-By-One Errors

```
for ( counter = 1; counter <= 10; counter++ ) {
   printf( "%u\n", counter );
}
```

- Notice that it uses the loop-continuation condition `counter <= 10`.
- If you incorrectly wrote `counter < 10`, then the loop would be executed only 9 times.
- This is a common logic error called an off-by-one error.

© By Pearson Education, Inc. All Rights Reserved.

Expressions in the for Statement's Header Are Optional

- The general format of the `for` statement is

```
for ( expression1; expression2; expression3 ) {
  statement
}
```

- If *expression2* is omitted, C assumes that the condition is true, thus creating an infinite loop.
- You may omit *expression1* if the control variable is initialized elsewhere in the program.
- *expression3* may be omitted if the increment is calculated by statements in the body of the `for` statement or if no increment is needed.

© By Pearson Education, Inc. All Rights Reserved.

for Statement: Notes and Observations

- The initialization, loop-continuation condition and increment can contain arithmetic expressions. For example,

- if x = 2 and y = 10, the statement

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to the statement

```
for ( j = 2; j <= 80; j += 5 )
```

- The "increment" may be negative (in which case it's really a decrement and the loop actually counts downward).

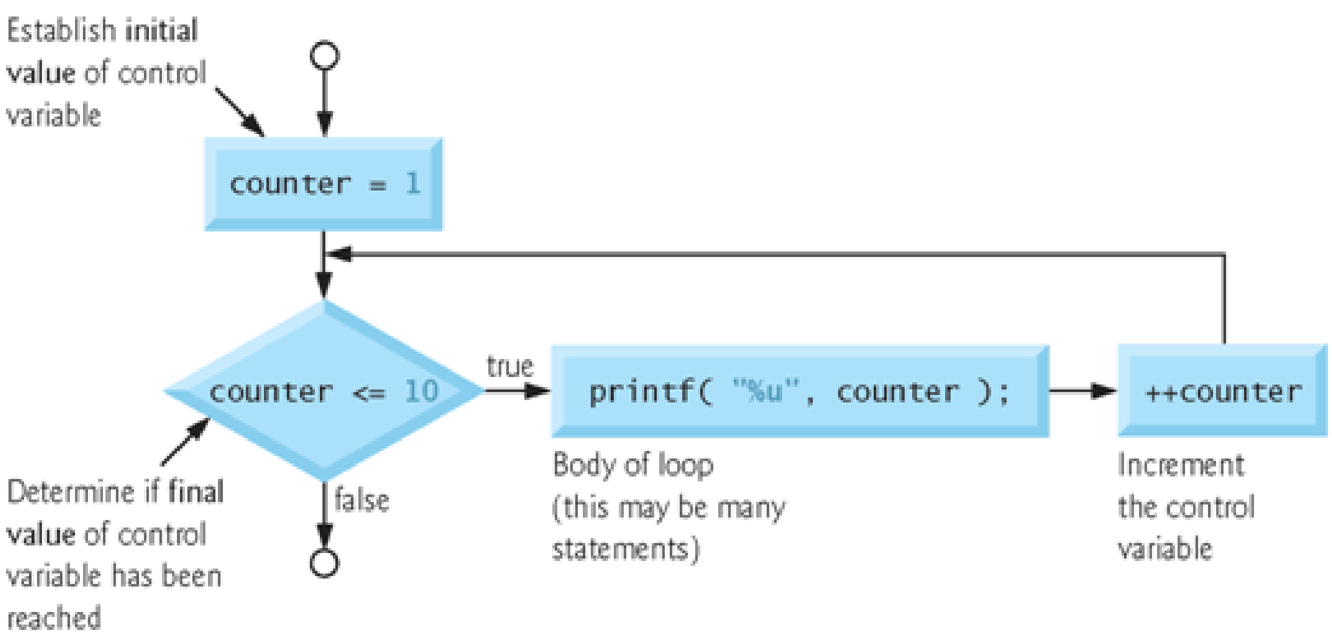
- If the loop-continuation condition is initially false, the loop body does not execute. Instead, execution proceeds with the statement following the `for` statement.

© By Pearson Education, Inc. All Rights Reserved.

for Statement: Notes and Observations (cont.)

```
for ( counter = 1; counter <= 10; ++counter )
   printf( "%u", counter );
```

- This flowchart makes it clear that the initialization occurs only once and that incrementing occurs *after* the body statement is performed.



Examples Using the for Statement

- Vary the control variable from 1 to 100 in increments of 1.
  `for ( i = 1; i <= 100; i++ )`

- Vary the control variable from 100 to 1 in decrements of 1.
  `for ( i = 100; i >= 1; --i )`

- Vary the control variable from 7 to 77 in steps of 7.
  `for ( i = 7; i <= 77; i += 7 )`

- Vary the control variable from 20 to 2 in steps of -2.
  `for ( i = 20; i >= 2; i -= 2 )`

- Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.
  `for ( j = 2; j <= 17; j += 3 )`

© By Pearson Education, Inc. All Rights Reserved.

Summing the Even Integers from 2 to 100

```c
// Fig. 4.5: fig04_05.c
// Summation with for.
#include <stdio.h>

// function main begins program execution
int main( void )
{
    unsigned int sum = 0; // initialize sum
    unsigned int number; // number to be added to sum

    for ( number = 2; number <= 100; number += 2 ) {
        sum += number; // add number to sum
    } // end for

    printf( "Sum is %u\n", sum ); // output sum
} // end function main
```

```
Sum is 2550
```

Fig. 4.5 | Summation with for.

for Statement: Notes and Observations

- The body of the `for` statement in Fig. 4.5 could actually be merged into the rightmost portion of the `for` header by using the comma operator as follows:

```
for ( number = 2; number <= 100; sum += number, number += 2 )
    ; // empty statement //
```

- The initialization `sum = 0` could also be merged into the initialization section of the `for`.

© By Pearson Education, Inc. All Rights Reserved.

Application: Compound-Interest Calculations

- Type `double` is a floating-point type like `float`, but a variable of type `double` can store a value of *much greater magnitude* with *greater precision* than `float`.
- The header `<math.h>` (line 4) should be included whenever a math function such as `pow` is used.
- Actually, this program would malfunction without the inclusion of `math.h`, as the linker would be unable to find the `pow` function.

© By Pearson Education, Inc. All Rights Reserved.

Application: Compound-Interest Calculations

- Consider the following problem statement:
  - A person invests $1000.00 in a savings account yielding 5% interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

$$a = p(1 + r)^n$$

where

p is the original amount invested (i.e., the principal)
r is the annual interest rate
n is the number of years
a is the amount on deposit at the end of the n^{th} year.

© By Pearson Education, Inc. All Rights Reserved.

```c
// Fig. 4.6: fig04_06.c
// Calculating compound interest.
#include <stdio.h>
#include <math.h>

// function main begins program execution
int main( void )
{
    double amount; // amount on deposit
    double principal = 1000.0; // starting principal
    double rate = .05; // annual interest rate
    unsigned int year; // year counter

    // output table column heads
    printf( "%4s%21s\n", "Year", "Amount on deposit" );

```

Fig. 4.6 | Calculating compound interest. (Part 1 of 2.)

© By Pearson Education, Inc. All Rights Reserved.

Application: Compound-Interest Calculations

- This problem involves a loop that performs the indicated calculation for each of the 10 years the money remains on deposit.

- $a = p(1 + r)^n$

```c
    // calculate amount on deposit for each of ten years
    for ( year = 1; year <= 10; ++year ) {

        // calculate new amount for specified year
        amount = principal * pow( 1.0 + rate, year );

        // output one table row
        printf( "%4u%21.2f\n", year, amount );
    } // end for
} // end function main
```

```
Year     Amount on deposit
   1               1050.00
   2               1102.50
   3               1157.63
   4               1215.51
   5               1276.28
   6               1340.10
   7               1407.10
   8               1477.46
   9               1551.33
  10               1628.89
```

Fig. 4.6 | Calculating compound interest. (Part 2 of 2.)

Application: Compound-Interest Calculations

- The `for` statement executes the body of the loop 10 times, varying a control variable from 1 to 10 in increments of 1.
- Although C does not include an exponentiation operator, we can use the Standard Library function `pow` for this purpose.
- The function `pow(x, y)` calculates the value of `x` raised to the `y`th power.
- It takes two arguments of type double and returns a `double` value.

© By Pearson Education, Inc. All Rights Reserved.

Formatting Numeric Output

- The conversion specifier `%21.2f` is used to print the value of the variable `amount` in the program.
- The `21` in the conversion specifier denotes the *field width* in which the value will be printed.
- A field width of `21` specifies that the value printed will appear in `21` print positions.
- The `2` specifies the *precision* (i.e., the number of decimal positions).

© By Pearson Education, Inc. All Rights Reserved.

Formatting Numeric Output

- If the number of characters displayed is less than the field width, then the value will automatically be *right justified* in the field.

- This is particularly useful for aligning floating-point values with the same precision (so that their decimal points align vertically).

- To *left justify* a value in a field, place a `-` (minus sign) between the `%` and the field width.

- The minus sign may also be used to left justify integers (such as in `%-6d`) and character strings (such as in `%-8s`).

© By Pearson Education, Inc. All Rights Reserved.

Type this program in codeblocks

```
#include <stdio.h>

int main()
{
    int num;

    printf("numbers:\t");

    for ( num = 3 ; num <= 23 ; num += 5 ) {
        printf("%d\t", num);
    }

    printf("\n\n num after for: %d\n", num);

    return 0;
}
```

© By Pearson Education, Inc. All Rights Reserved.

SRE1 - for loop

- Write a program that uses a for statement to print the odd integers from 1 to 21.
- Your program must print " : m5" in front of the numbers in the list that are multiples of 5.
- 1
- 3
- 5 : m5
- 7
- …

© By Pearson Education, Inc. All Rights Reserved.

SRE2 – Quiz preparation

- In mathematics, the factorial of a positive integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example,
- 5! = 5 x 4 x 3 x 2 x 1 = 120
- The value of 0! is 1
- Write a program that uses a for statement to calculate and print the factorial of a positive integer number.

© By Pearson Education, Inc. All Rights Reserved.

`switch` Multiple-Selection Statement

- Occasionally, an algorithm will contain a *series of decisions* in which a variable or expression is tested separately for each of the constant integral values it may assume, and different actions are taken.

- This is called *multiple selection*.

- C provides the `switch` multiple-selection statement to handle such decision making.

© By Pearson Education, Inc. All Rights Reserved.

`switch` Multiple-Selection Statement

- The `switch` statement consists of a series of `case` labels, an optional `default` case and statements to execute for each case.
- Figure 4.7 uses `switch` to count the number of each different letter grade students earned on an exam.

© By Pearson Education, Inc. All Rights Reserved.

```c
// Fig. 4.7: fig04_07.c
// Counting letter grades with switch.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int grade; // one grade
   unsigned int aCount = 0; // number of As
   unsigned int bCount = 0; // number of Bs
   unsigned int cCount = 0; // number of Cs
   unsigned int dCount = 0; // number of Ds
   unsigned int fCount = 0; // number of Fs

   puts( "Enter the letter grades."  );
   puts( "Enter the EOF character to end input."  );

   // loop until user types end-of-file key sequence
   while ( ( grade = getchar() ) != EOF ) {

      // determine which grade was input
      switch ( grade ) { // switch nested in while

```

Fig. 4.7 | Counting letter grades with switch. (Part 1 of 4.)

© By Pearson Education, Inc. All Rights Reserved.

Reading Character Input

- In the program, the user enters letter grades for a class.
- In the `while` header (line 19),
  - `while ( ( grade = getchar() ) != EOF )`
- the parenthesized assignment `(grade = getchar())` executes first.
- The `getchar` function (from `<stdio.h>`) reads one character from the keyboard and stores that character in the integer variable `grade`.
- An important feature of C is that characters can be stored in any integer data type because they're represented as one-byte integers in the computer.

© By Pearson Education, Inc. All Rights Reserved.

```
// determine which grade was input
switch ( grade ) { // switch nested in while

   case 'A': // grade was uppercase A
   case 'a': // or lowercase a
      ++aCount; // increment aCount
      break; // necessary to exit switch

   case 'B': // grade was uppercase B
   case 'b': // or lowercase b
      ++bCount; // increment bCount
      break; // exit switch

   case 'C': // grade was uppercase C
   case 'c': // or lowercase c
      ++cCount; // increment cCount
      break; // exit switch

   case 'D': // grade was uppercase D
   case 'd': // or lowercase d
      ++dCount; // increment dCount
      break; // exit switch

   case 'F': // grade was uppercase F
   case 'f': // or lowercase f
      ++fCount; // increment fCount
      break; // exit switch

```

© By Pearson Education, Inc. All Rights Reserved.

```
            case '\n': // ignore newlines,
            case '\t': // tabs,
            case ' ': // and spaces in input
               break; // exit switch

            default: // catch all other characters
               printf( "%s", "Incorrect letter grade entered." );
               puts( " Enter a new grade." );
               break; // optional; will exit switch anyway
         } // end switch
      } // end while

      // output summary of results
      puts( "\nTotals for each letter grade are:" );
      printf( "A: %u\n", aCount ); // display number of A grades
      printf( "B: %u\n", bCount ); // display number of B grades
      printf( "C: %u\n", cCount ); // display number of C grades
      printf( "D: %u\n", dCount ); // display number of D grades
      printf( "F: %u\n", fCount ); // display number of F grades
   } // end function main
```

Fig. 4.7 | Counting letter grades with switch. (Part 3 of 4.)

© By Pearson Education, Inc. All Rights Reserved.

```
Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z
```

Not all systems display a representation of the EOF character

```
Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Fig. 4.7 | Counting letter grades with `switch`. (Part 4 of 4.)

© By Pearson Education, Inc. All Rights Reserved.

`switch` Statement details

- Keyword `switch` is followed by the variable name `grade` in parentheses.
  - `switch ( grade )`
- This is called the controlling expression.

- The value of this expression is compared with each of the `case` labels.

- Assume the user has entered the letter `C` as a grade.
- `C` is automatically compared to each `case` in the `switch`.
- If a match occurs (`case 'C':`), the statements for that `case` are executed.

© By Pearson Education, Inc. All Rights Reserved.

`switch` Statement details

- In the case of the letter 'c', `cCount` is incremented by 1, and the `switch` statement is exited immediately with the `break` statement.
  - case 'c':
  - ++cCount;
  - break;

- The `break` statement causes program control to continue with the first statement after the `switch` statement.

- If `break` is not used anywhere in a `switch` statement, then each time a match occurs in the statement, the statements for all the remaining `case`s will be executed.

© By Pearson Education, Inc. All Rights Reserved.

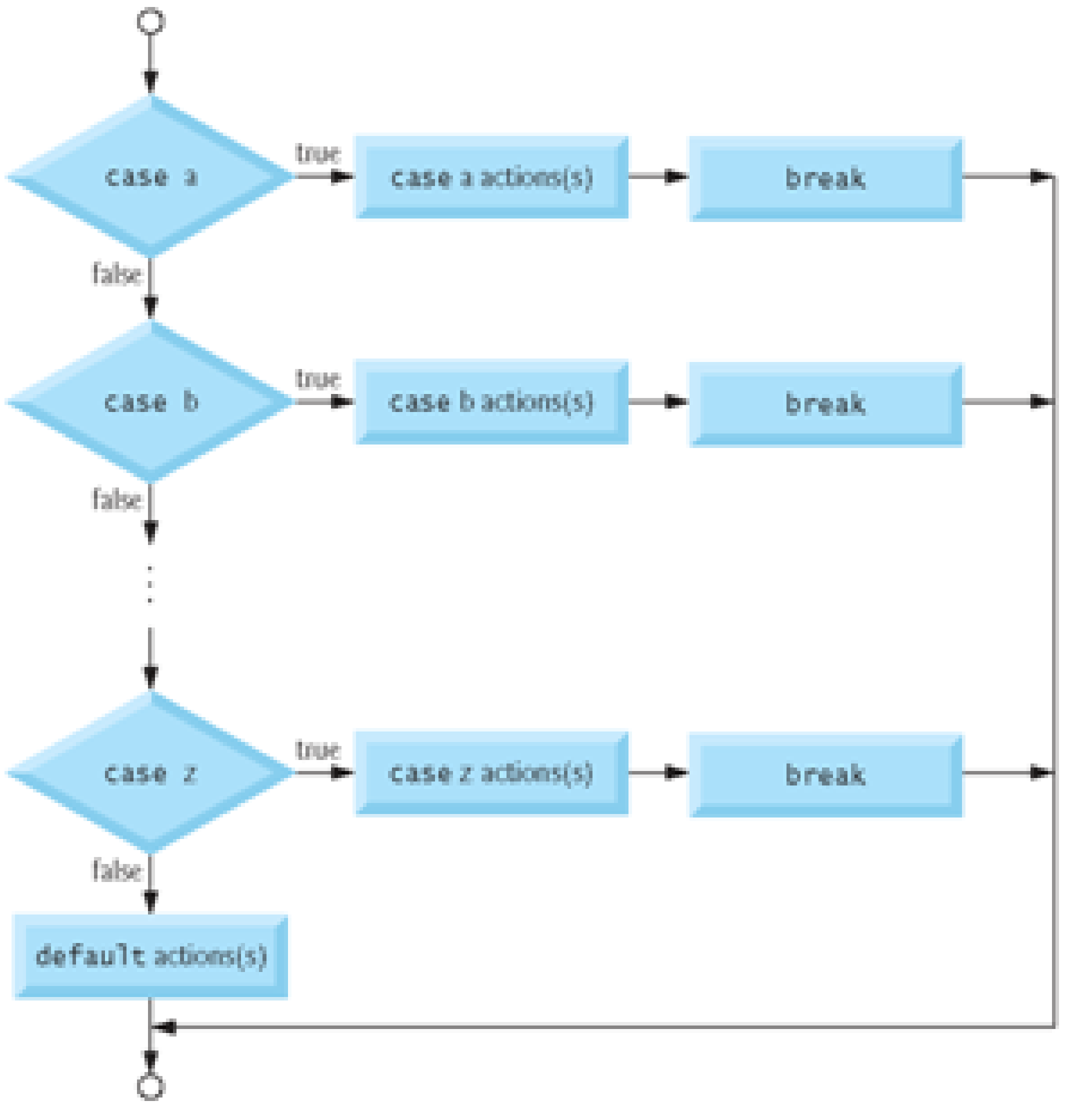


Fig. 4.8 | `switch` multiple-selection statement with `break`s.

© By Pearson Education, Inc. All Rights Reserved.

Ignoring Newline, Tab and Blank Characters in Input

- In the `switch` statement of Fig. 4.7, the lines

  ```
  case '\n': // ignore newlines,
  case '\t': // tabs,
  case ' ': // and spaces in input
     break; // exit switch
  ```

  cause the program to skip newline, tab and blank characters.

- By including the preceding cases in our `switch` statement, we prevent the error message in the `default` case from being printed each time a newline, tab or space is encountered in the input.

© By Pearson Education, Inc. All Rights Reserved.

`switch` Multiple-Selection Statement (Cont.)

- Listing several case labels together (such as `case 'D': case 'd':`) simply means that the *same* set of actions is to occur for either of these cases.
- A character constant can be represented as the specific character in single quotes, such as `'A'`.
- Characters *must* be enclosed within single quotes to be recognized as character constants—characters in double quotes are recognized as strings.
- Integer constants are simply integer values.

© By Pearson Education, Inc. All Rights Reserved.

Notes on Integral Types

- Different applications may need integers of different sizes.
- C provides several data types to represent integers.
- In addition to `int` and `char`, C provides types `short` and `long`
- The C standard specifies the minimum range of values for each integer type, but the actual range may be greater and depends on the implementation.

© By Pearson Education, Inc. All Rights Reserved.

Notes on Integral Types

- For `short int`s the minimum range is
  - –32767 to +32767
- The minimum range of values for `long` is
  - –2147483647 to +2147483647
- The data type `unsigned char` can be used to represent integers in the range 0 to 255 or any of the characters in the computer's character set.

© By Pearson Education, Inc. All Rights Reserved.

Type this program in codeblocks

```c
#include <stdio.h>

int main()
{
    int num;

    printf("Give me a number: ");
    scanf("%d", &num);

    switch (num%2) {
    case 0:
        printf("%d is even\n", num);
        break;
    case 1:
        printf("%d is odd\n", num);
        break;
    default:
        printf("Don't forget this case\n");
        break;
    }

    return 0;
}
```

© By Pearson Education, Inc. All Rights Reserved.

Self Review Exercise - switch

Exercise 4.19 – Calculating sales

- Write a program that reads a series of pairs of numbers as follows:
  - Product Number
  - Quantity
- Your program must use a switch statement to help determine the retail price for each product
- Assume product number -1 as the sentinel value
- Note: For the product numbers and retail prices use the table on page 153

© By Pearson Education, Inc. All Rights Reserved.