Chapter 3: Expressions and Interactivity

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Topics

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Topics (continued)

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3.1 The `cin` Object

- Standard input object
- Like **`cout`**, requires **`iostream`** file
- Used to read input from keyboard
- Often used with **`cout`** to display a user prompt first
- Data is retrieved from **`cin`** with **>>**
- Input data is stored in one or more variables

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

The `cin` Object

- **cin** can be used to read data typed at the keyboard.

```
int height;
cout << "How tall is the room? ";
cin  >> height;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
// This program calculates and displays the area of a rectangle.
#include <iostream>
using namespace std;

int main()
{
   int length, width, area;

   cout << "This program calculates the area of a rectangle.\n";

    // Have the user input the rectangle's length and width
   cout << "What is the length of the rectangle? ";
   cin >> length;
   cout << "What is the width of the rectangle? ";
   cin >> width;

   // Compute and display the area
   area = length * width;
   cout << "The area of the rectangle is " << area << endl;
   return 0;
}
```

Prompt

```
This program calculates the area of a rectangle.
What is the length of the rectangle? 10[Enter]
What is the width of the rectangle? 20[Enter]
The area of the rectangle is 200.
```

3.1 What header file must be included in programs using cin?

3.3 Where does cin read its input from?

3.4 True or False: cin requires the user to press the [Enter] key after entering data.

Complete the following program skeleton so it asks for the user’s weight (in pounds) and displays the equivalent weight in kilograms.

```
int main()
{
    double pounds, kilograms;
    // Write a prompt to tell the user to enter his or her weight
    // in pounds.
    // Write code here that reads in the user's weight in pounds.


    // The following line does the conversion.
    kilograms = pounds / 2.2;

    // Write code here that displays the user's weight in kilograms.

    return 0;
}
```

Enter your weight in pounds: **140[Enter]**

Enter your weight in pounds:
140[Enter]

3.2 Mathematical Expressions

- An expression can be a constant, a variable, or a combination of constants and variables combined with operators
- Examples of mathematical expressions:

```
 2
height
a + b / c
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Using Mathematical Expressions

- Can be used in assignment statements, with `cout`, and in other types of statements

These are expressions

```
area = 2 * PI * radius;
cout << "border is: " << (2*(l+w));
```

It is not good practice to perform mathematical operations within a cout statement

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Order of Operations

- In an expression with > 1 operator, evaluate in this order

Do first: `-` (unary negation) in order, left to right

Do next: `* / %` in order, left to right

Do last: `+ -` in order, left to right

In the expression `2 + 2 * 2 – 2` ,

Evaluate 2nd **Evaluate 1st** **Evaluate 3rd**

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Associativity of Operators

- `-` (unary negation) associates right to left
- `* / % + -` all associate left to right
- parentheses `( )` can be used to override the order of operations

```
 2 + 2  *  2 – 2  = 4
(2 + 2) *  2 – 2  = 6
 2 + 2  * (2 – 2) = 2
(2 + 2) * (2 – 2) = 0
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Algebraic Expressions

- Multiplication requires an operator

  Area = lw is written as `Area = l * w;`

- There is no exponentiation operator

  Area = s² is written as `Area = pow(s, 2);`

  (note: **pow** requires the **cmath** header file)

- Parentheses may be needed to maintain order of operations

  $m = \frac{y_2 - y_1}{x_2 - x_1}$ is written as

  `m = (y2-y1)/(x2-x1);`

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Grouping with Parentheses

average = (a + b) / 4;

Without the parentheses

average = a + b / 4;

b would be divided by 4 **before** adding a to the result.

```
// This program calculates the area of a circle. The formula for the
// area of a circle is PI times the radius squared. PI is 3.14159.
#include <iostream>
#include <cmath>          // Needed for the pow function
using namespace std;

int main()
{
    double area, radius;
    cout << "This program calculates the area of a circle.\n";

    // Get the radius
    cout << "What is the radius of the circle? ";
    cin >> radius;

    // Compute and display the area
    area = 3.14159 * pow(radius, 2);        // πr²
    cout << "The area is " << area << endl;
    return 0;
}
```

3.11 Write C++ expressions for the following algebraic expressions:

y = 6x   a = 2 b + 4c $y = x^3$

$z = \frac{x + 2}{z^2}$ $y = \frac{x^2}{z^2}$

volume = $\pi r^2 h$ where π is 3.14159

3.3 Implicit Type Conversion

- Operations are performed between operands of the same type
- If not of the same type, C++ will automatically convert one to be the type of the other
- This can impact the results of calculations

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Hierarchy of Data Types

- Highest `long double`

  `double`

  `float`

  `unsigned long`

  `long`

  `unsigned int`

  `int`

  `unsigned short`

  `short`

- Lowest `char`

- Ranked by largest number they can hold

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Rules

2) When operating on values of different data types, the lower one is promoted to the type of the higher one.

3) When using the **=** operator, the type of expression on right will be converted to the type of variable on left

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Rule 2: When an operator works with two values of different data types, the lower-ranking value is promoted to the type of the higher-ranking value.

```
int years;
double interestRate, interest;
…
…
interest = years * interestRate;
```

Before the multiplication takes place, the value in years will be promoted to a double.

Rule 3: When the final value of an expression is assigned to a variable, it will be converted to the data type of that variable.

```
int x =2;
double y = 3.75;
x = y;                    // x is assigned 3
                          // y remains 3.75
                          // decimal value truncated
```

```
int x =2;
double y = 3.75;
y = x;                    // y is assigned 2.0
                          // x remains 2
                          //no harm as 2 and 2.0 are equivalent
```

3.4 Explicit Type Conversion

- Also called type casting
- Allows you to perform manual data type conversion
- Format

```
static_cast<type>(expression)
```

```
double number = 3.7;
int val;
val = static_cast<int>(number);
```

↓

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Type cast expressions are useful in situations where C++ will not perform the desired conversion automatically.

```
double avg;
int count=4, sum=10;
…
avg = sum / count;
cout << avg;                             // 2.0     incorrect !

avg = static_cast<double>(sum)/count;
cout << avg;                             // 2.5

avg = sum/ static_cast<double>(count);
cout << avg;                             // 2.5
```

↓

Older type cast styles

```
double avg;
int count=4, sum=10;
…

avg = static_cast<double>(sum)/count;
cout << avg;                             // 2.5

avg = (double)sum /count;           // prefix notation
cout << avg;                             // 2.5

avg = double(sum) /count;           // functional  notation
cout << avg;                             // 2.5
```

3.14 Assume the following variable definitions:

int a = 5, b = 12;
double x = 3.4, z = 9.1;

What are the values of the following expressions?

A) b / a

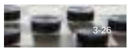
B) x * a

C) static_cast<double>(b / a)

D) static_cast<double>(b) / a

HW
Exercises_Chapter 3
Programming Challenges
1. Miles per Gallon
Write a program that calculates a car's gas mileage. The program should ask the user to enter the number of gallons of gas the car can hold and the number of miles it can be driven on a full tank. It should then calculate and display the number of miles per gallon the car gets.

2. Stadium Seating
There are three seating categories at a stadium. For a softball game, Class A seats cost $15, Class B seats cost $12, and Class C seats cost $9. Write a program that asks how many tickets for each class of seats were sold, then displays the amount of income generated from ticket sales.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley


3.5 Overflow and Underflow

- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- Just as a bucket will overflow if you try to put more water in it than it can hold.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Overflow Example

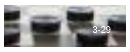
```
// Create a short int initialized to
// the largest value it can hold
short int num = 32767;

cout << num;      // Displays 32767
num = num + 1;
cout << num;      // Displays -32768
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Handling Overflow and Underflow

Different systems handle the problem differently. They may

- display a warning / error message
- display a dialog box and ask what to do
- stop the program
- continue execution with the incorrect value

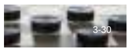
Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley


3.6 Named Constants

- Also called constant variables
- Variables whose content cannot be changed during program execution
- Used for representing constant values with descriptive names

```
const double TAX_RATE = 0.0675;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley


When a named constant is defined it must be initialized with a value. It cannot be defined and then later assigned a value with an assignment statement.

```
const double INTEREST_RATE;              // illegal
INTEREST_RATE = 0.129;                   // illegal
```

```
const double INTEREST_RATE = 0.129;
```

If the named constant INTEREST_RATE has been correctly defined, the program statement

```
newAmount = balance * 0.129;
```

can be changed to read

```
newAmount = balance * INTEREST_RATE;
```

Program 3-11

```
// This program calculates the area of a circle..
#include <iostream>
#include <cmath>              // Needed for the pow function
using namespace std;

int main()
{
    const double PI = 3.14159;           // PI is a named constant
    double area, radius;

    cout << "This program calculates the area of a circle.\n";

    // Get the radius
    cout << "What is the radius of the circle? ";
    cin >> radius;

    // Compute and display the area
    area = PI * pow(radius, 2);
    cout << "The area is " << area << endl;
    return 0;
}
```

`const` vs. `#define`

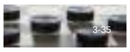
SKIP

`#define`
- C-style of naming constants
  `#define NUM_STATES 50`
  no ; goes here
- Interpreted by pre-processor rather than compiler
- Does not occupy a memory location like a constant variable defined with `const`
- Instead, causes a text substitution to occur. In above example, every occurrence in program of `NUM_STATES` will be replaced by `50`

3.7 Multiple and Combined Assignment

SKIP

- The assignment operator (=) can be used more than 1 time in an expression
  `x = y = z = 5;`
- Associates right to left
  `x = (y = (z = 5));`
  Done 3rd   Done 2nd   Done 1st



Quite often programs have assignment statements of the following form:

```
number = number + 1;
```

We first evaluate the expression on the right hand side

```
number + 1
```

Then we assign its value to the variable on the left hand side

number ← umber + 1

Effectively, this statement adds 1 to number

In a similar fashion, the following statement subtracts 5 from number.

number = number – 5;

If you have never seen this type of statement before, it might cause some initial confusion because the same variable name appears on both sides of the assignment operator.

Assume x = 6

Statement	What It Does	Value of x After the Statement
x = x + 4;	Adds 4 to x	10
x = x - 3;	Subtracts 3 from x	3
x = x * 10;	Multiplies x by 10	60
x = x / 2;	Divides x by 2	3
x = x % 4	Makes x the remainder of x / 4	2

Because these types of operations are so common in programming, C++ offers a special set of operators designed specifically for these jobs.

↓

Combined Assignment

- Applies an arithmetic operation to a variable and assigns the result as the new value of that variable
- Operators: `+=   -=   *=   /=   %=`
- Example:
- `sum += amt;` is short for `sum = sum + amt;`

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

More Examples

`x += 5;`	means	`x = x + 5;`
`x -= 5;`	means	`x = x – 5;`
`x *= 5;`	means	`x = x * 5;`
`x /= 5;`	means	`x = x / 5;`
`x %= 5;`	means	`x = x % 5;`

The right hand side is evaluated before the combined assignment operation is done.

`x *= a + b;` means `x = x * (a + b);`

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Program 3-13

```
// This program tracks the inventory of two widget stores.
// It illustrates the use of multiple and combined assignment.
#include <iostream>
using namespace std;

int main()
{
    int begInv, // Beginning inventory for both stores
    sold,   // Number of widgets sold
    store1,          // Store 1's inventory
    store2;          // Store 2's inventory

    // Get the beginning inventory for the two stores
    cout << "One week ago, 2 new widget stores opened\n";
    cout << "at the same time with the same beginning\n";
    cout << "inventory. What was the beginning inventory? ";
    cin >> begInv;
```

← continued

```
    // Set each store's inventory
    store1 =  begInv;
    store2 =  begInv;

    // Get the number of widgets sold at each store
    cout << "How many widgets has store 1 sold? ";
    cin >> sold;
    store1 -= sold; // Adjust store 1's inventory

    cout << "How many widgets has store 2 sold? ";
    cin >> sold;
    store2 -= sold; // Adjust store 2's inventory

    // Display each store's current inventory
    cout << "\nThe current inventory of each store:\n";
    cout << "Store 1: " << store1 << endl;
    cout << "Store 2: " << store2 << endl;
    return 0;
}
```

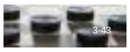
```
One week ago, 2 new widget stores opened
at the same time with the same beginning
inventory. What was the beginning inventory? 100[Enter]
How many widgets has store 1 sold? 25[Enter]
How many widgets has store 2 sold? 15[Enter]
The current inventory of each store:
Store 1: 75
Store 2: 85
```

↓

3.8 Formatting Output

- Can control how output displays for numeric and string data
  - size
  - position
  - number of digits
- Requires **`iomanip`** header file

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-14

```
// This program displays three rows of numbers.
#include <iostream>
using namespace std;
int main()
{
    int num1 = 2897, num2 = 5, num3 = 837,
    num4 = 34, num5 = 7, num6 = 1623,
    num7 = 390, num8 = 3456, num9 = 12;

    // Display the first row of numbers
    cout << num1 << " " << num2 << " " << num3 << endl;

    // Display the second row of numbers
    cout << num4 << " " << num5 << " " << num6 << endl;

    // Display the third row of numbers
    cout << num7 << " " << num8 << " " << num9 << endl;

    return 0;
}
```

```
2897 5 837
34 7 1623
390 3456 12
```

Stream Manipulators

- Used to control features of an output field
- Some affect just the next value displayed
  - **`setw(x)`**: Print in a field at least **`x`** spaces wide. Use more spaces if specified field width is not big enough.

```
value = 23;
cout << setw(5) << value;
```

```
   23
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Program 3-15

```
// This program uses setw to display three rows of numbers so they align.
#include <iostream>
#include <iomanip> // Header file needed to use setw
using namespace std;

int main()
{
    int num1 = 2897, num2 = 5, num3 = 837,
    num4 = 34, num5 = 7, num6 = 1623,
    num7 = 390, num8 = 3456, num9 = 12;

    // Display the first row of numbers
    cout << setw(6) << num1 << setw(6) << num2 << setw(6) << num3 << endl;

    // Display the second row of numbers
    cout << setw(6) << num4 << setw(6) << num5 << setw(6) << num6 << endl;

    // Display the third row of numbers
    cout << setw(6) << num7 << setw(6) << num8 << setw(6) << num9 << endl;
    return 0;
}
```

```
  2897     5   837
    34     7  1623
   390  3456    12
```

The setprecision Manipulator

Floating-point values may be rounded to a number of *significant digits*, or *precision*, which is the total number of digits that appear before and after the decimal point.

Number	Manipulator	Value Displayed
28.92786	setprecision(3)	28.9
21.40	setprecision(5)	21.4
109.50	setprecision(4)	109.5
34.78596	setprecision(2)	35

Program 3-17

```
// This program demonstrates how the setprecision manipulator
// affects the way a floating-point value is displayed.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double, number1 = 132.364, number2 = 26.91;
    double quotient = number1 / number2;

    cout << quotient << endl;
    cout << setprecision(5) << quotient << endl;
    cout << setprecision(4) << quotient << endl;
    cout << setprecision(3) << quotient << endl;
    cout << setprecision(2) << quotient << endl;
    cout << setprecision(1) << quotient << endl;
    return 0;
}
```

```
4.91877
4.9188
4.919
4.92
4.9
5
```

Stream Manipulators

- Some affect values until changed again
  - `fixed`: Use decimal notation (not E-notation) for floating-point values.
  - `setprecision(x)`:
    - When used with `fixed`, print floating-point value using `x` digits after the decimal.
    - Without `fixed`, print floating-point value using `x` significant digits.
  - `showpoint`: Always print decimal for floating-point values.
  - `left, right`: left-, right justification of value

Manipulator Examples

```
const float e = 2.718;
float price = 18.0;
cout << setw(8) << e << endl;
cout << left << setw(8) << e
     << endl;
cout << setprecision(2);
cout << e << endl;
cout << fixed << e << endl;
cout << setw(6) << price;
```

Displays
```
^^^2.718

2.718^^^

2.7
2.72
^18.00
```

3.9 Working with Characters and String Objects

- `char`: holds a single character

    char letter1 = 'A',
          letter2 = 'B';

- `string`: holds a sequence of characters

    string name1 = "Mark Twain",
           name2 = "Samuel James";

- Both can be used in assignment statements

    letter2 = letter1;   // Now letter2's value is 'A'
    name2 = name1;   // Now name2's value is "Mark Twain"

- Both can be displayed with `cout` and `<<`

    cout << letter1 << ". " << name1 << endl;

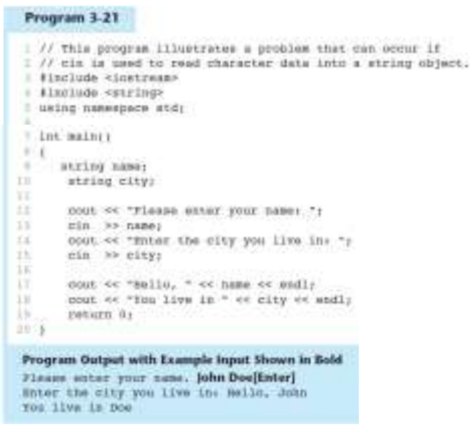
    A. Mark Twain

String Input

When cin reads data it passes over and ignores any leading *whitespace* characters (spaces, tabs, or line breaks). However, once it comes to the first nonblank character and starts reading, it stops reading when it gets to the next whitespace character.

If we use the following statement
    cin >> name1;

we can input "Mark", or even " Mark", but not "Mark Twain" because cin cannot input strings that contain embedded spaces.


Program 3-21

```
// This program illustrates a problem that can occur if
// cin is used to read character data into a string object.
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string name;
   string city;

   cout << "Please enter your name: ";
   cin  >> name;
   cout << "Enter the city you live in: ";
   cin  >> city;

   cout << "Hello, " << name << endl;
   cout << "You live in " << city << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

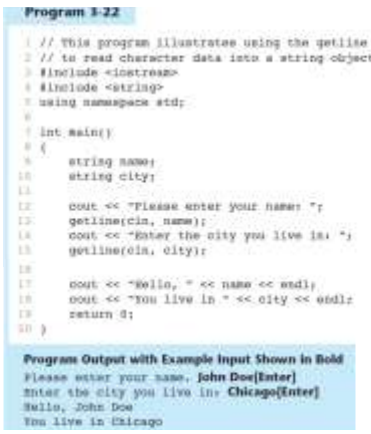
```
Please enter your name: John Doe[Enter]
Enter the city you live in: Hello, John
You live in Doe
```

To solve this problem, C++ provides a special function called *getline*. This function will read in an entire line, including leading and embedded spaces, and store it in a string object.

The getline function looks like the following, where cin is the input stream we are reading from and inputLine is the name of the string variable receiving the input string.

```
getline(cin, inputLine);
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-22

```
// This program illustrates using the getline function
// to read character data into a string object.
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string name;
    string city;

    cout << "Please enter your name: ";
    getline(cin, name);
    cout << "Enter the city you live in: ";
    getline(cin, city);

    cout << "Hello, " << name << endl;
    cout << "You live in " << city << endl;
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Please enter your name: John Doe[Enter]
Enter the city you live in: Chicago[Enter]
Hello, John Doe
You live in Chicago
```

String Input

Reading in a string object

```
string str;

cin >> str;             // Reads in a string
                        // with no blanks

getline(cin, str);      // Reads in a string
                        // that may contain
                        // blanks
```

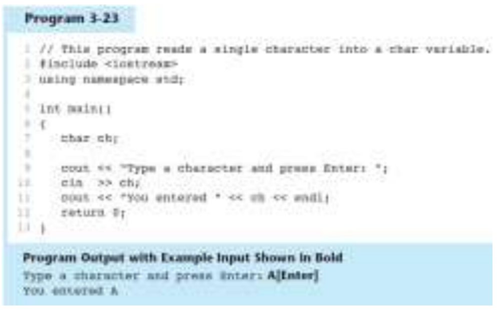
Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Character Input

Reading in a character

```
char ch;

cin >> ch;     // Reads in any non-blank char
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-23

```
// This program reads a single character into a char variable.
#include <iostream>
using namespace std;

int main()
{
   char ch;

   cout << "Type a character and press Enter: ";
   cin  >> ch;
   cout << "You entered " << ch << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Type a character and press Enter: A[Enter]
You entered A
```

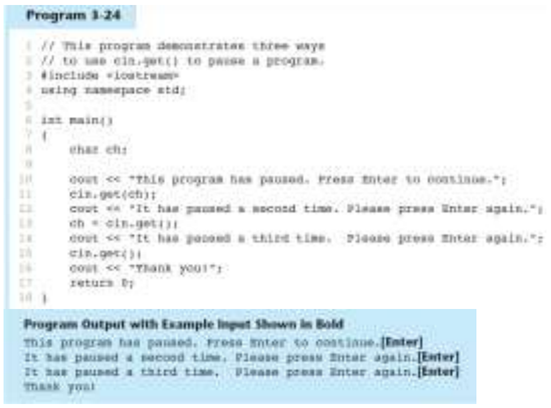
Using cin.get

As with string input, however, there are times when using cin >> to read a character does not do what we want. For example, because it passes over all leading whitespace, it is impossible to input just a blank or [Enter] with cin >>.

`cin.get(ch);` // Reads in *any* char and saves it in ch

`ch = cin.get();` // Reads in *any* char and saves it in ch

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-24

```
// This program demonstrates three ways
// to use cin.get() to pause a program.
#include <iostream>
using namespace std;

int main()
{
    char ch;

    cout << "This program has paused. Press Enter to continue.";
    cin.get(ch);
    cout << "It has paused a second time. Please press Enter again.";
    ch = cin.get();
    cout << "It has paused a third time.  Please press Enter again.";
    cin.get();
    cout << "Thank you!";
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
This program has paused. Press Enter to continue.[Enter]
It has paused a second time. Please press Enter again.[Enter]
It has paused a third time.  Please press Enter again.[Enter]
Thank you!
```

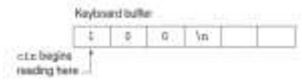
Mixing cin >> and cin.get

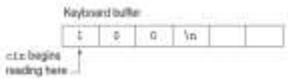
Mixing cin >> with cin.get can cause an annoying and hard-to-find problem.

```
1 char ch;                    // Define a character variable
2 int number;                 // Define an integer variable
3 cout << "Enter a number: ";
4 cin >> number;              // Read an integer
3 cout << "Enter a character: ";
6 ch = cin.get();             // Read a character
7 cout << "Thank You!\n";
```

These statements allow the user to enter a number, but not a character. It will appear that the cin.get statement on line 6 has been skipped.

This happens because both cin >> and cin.get read the user's keystrokes from the keyboard buffer. After entering a number in response to the first prompt, the user presses the [Enter] key. Pressing this key causes a newline character ('\n') to be stored in the keyboard buffer. For example, suppose the user enters 100 and presses [Enter]. The input will be stored in the keyboard buffer as shown





When the cin >> statement reads data from the keyboard buffer, it stops reading at the newline character. In our example, 100 is read in and stored in the number variable. The newline character is left in the keyboard buffer. However, cin.get always reads the next character in the buffer, no matter what it is, without skipping over whitespace. It only waits for the user to input a value if the keyboard buffer is empty. When cin.get finds the newline character in the buffer, it uses it and does not wait for the user to input another value. You can remedy this situation by using the cin.ignore function

```
cin.ignore(); // Skips over next char in
              // the input buffer
```

The statements that mix cin >> and cin.get can be repaired by inserting a cin.ignore statement after the cin >> statement:

```
cout << "Enter a number: ";
cin >> number;
cin.ignore(); // Skip the newline character
cout << "Enter a character: ";
cin.get(ch);
cout << "Thank You!" << endl;
```

Character Input

Reading in a character

```
char ch;
cin >> ch;      // Reads in any non-blank char
cin.get(ch);    // Reads in any char
cin.ignore();   // Skips over next char in
                // the input buffer
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

String Operators

= Assigns a value to a string

```
string words;
words = "Tasty ";
```

+ Joins two strings together

```
string s1 = "hot", s2 = "dog";
string food = s1 + s2; // food = "hotdog"
```

+= Concatenates a string onto the end of another one

```
words += food; // words now = "Tasty hotdog"
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3.10 Using C-Strings

CONCEPT: C-strings provide another way to store and work with strings.

Because this was the way to create a string variable in C, a string defined in this manner is called a ***C-string***.

Here is a statement that defines word to be an array of characters that will hold a C-string and initializes it to "Hello".

char word[10] = "Hello";

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3.10 Using C-Strings

- C-string is stored as an array of characters
- Programmer must indicate maximum number of characters at definition

```
const int SIZE = 5;
char temp[SIZE] = "Hot";
```

- NULL character (`\0`) is placed after final character to mark the end of the string

- Programmer must make sure array is big enough for desired use; `temp` can hold up to 4 characters plus the `\0`.

H	o	t	\0	

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Because C-strings are harder to work with than string objects, you might be wondering why you are learning about them. There are two reasons.

- First, you are apt to encounter older programs that use them, so you need to understand them.
- Second, even though strings can now be declared as string objects in most cases, there are still times when only C-strings will work. You will be introduced to some of these cases later.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

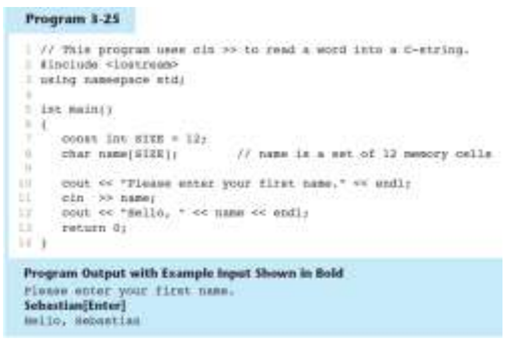
C-String Input

- Reading in a C-string

```
const int SIZE = 10;
char Cstr[SIZE];
cin >> Cstr;       // Reads in a C-string with no
                   // blanks. Will write past the
                   // end of the array if input string
                   // is too long.
cin.getline(Cstr, 10);
                   // Reads in a C-string that may
                   // contain blanks. Ensures that <= 9
                   // chars are read in.
```

- Can also use **`setw()`** and **`width()`** to control input field widths

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-25

```
// This program uses cin >> to read a word into a C-string.
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 12;
   char name[SIZE];         // name is a set of 12 memory cells

   cout << "Please enter your first name." << endl;
   cin  >> name;
   cout << "Hello, " << name << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Please enter your first name.
Sebastian[Enter]
Hello, Sebastian
```

C-String Initialization vs. Assignment

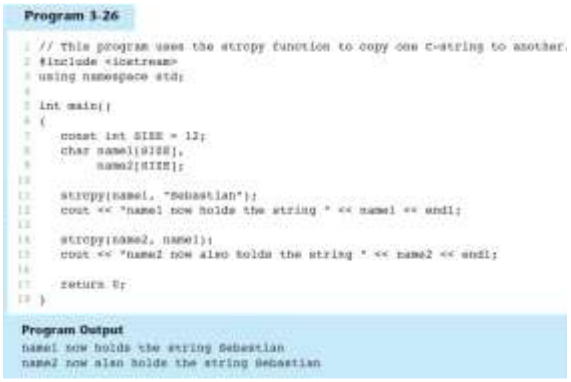
- A C-string can be initialized at the time of its creation, just like a string object

```
const int SIZE = 10;
char month[SIZE] = "April";
```

- However, a C-string cannot later be assigned a value using the = operator; you must use the **`strcpy()`** function

```
char month[SIZE];
month = "August"           // wrong!
strcpy(month, "August"); //correct
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-26

```
// This program uses the strcpy function to copy one C-string to another.
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 12;
   char name1[SIZE],
        name2[SIZE];

   strcpy(name1, "Sebastian");
   cout << "name1 now holds the string " << name1 << endl;

   strcpy(name2, name1);
   cout << "name2 now also holds the string " << name2 << endl;

   return 0;
}
```

Program Output

```
name1 now holds the string Sebastian
name2 now also holds the string Sebastian
```

3.11 More Mathematical Library Functions

- These require **`cmath`** header file
- Take **`double`** arguments and return a **`double`**
- Commonly used functions

`abs`	Absolute value
`sin`	Sine
`cos`	Cosine
`tan`	Tangent
`sqrt`	Square root
`log`	Natural (e) log

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Exercises Ch 3

Programming Challenges

6. Test Average

Write a program that asks for five test scores. The program should calculate the average test score and display it. The number displayed should be formatted in fixed-point notation, with one decimal point of precision.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

8. Box Office

A movie theater only keeps a percentage of the revenue earned from ticket sales. The remainder goes to the distributor. Write a program that calculates a theater's gross and net box office profit for a night. The program should ask for the name of the movie, and how many adult and child tickets were sold. (The price of an adult ticket is $6.00 and a child's ticket is $3.00.) It should display a report similar to the following:

Movie Name: "Wheels of Fury"
Adult Tickets Sold: 382
Child Tickets Sold: 127
Gross Box Office Profit: $ 2673.00
Amount Paid to Distributor: – $ 2138.40
Net Box Office Profit: $ 534.60

Assume the theater keeps 20 percent of the gross box office profit.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

HW

Lab 3

1. Students should read the Pre-lab Reading Assignment before coming to lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab. (photocopy or copy/paste, answer then print and bring to class.)

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley