Chapter 3: Expressions and Interactivity

Starting Out with C++
Early Objects
Eighth Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley
is an imprint of
PEARSON

Topics

3.1 The **`cin`** Object

3.2 Mathematical Expressions

3.3 Data Type Conversion and Type Casting

3.4 Overflow and Underflow

3.5 Named Constants

Topics (continued)

3.6 Multiple and Combined Assignment

3.7 Formatting Output

3.8 Working with Characters and Strings

3.9 Using C-Strings

3.10 More Mathematical Library Functions

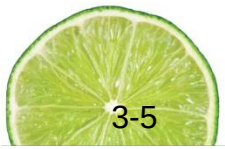
3.1 The `cin` Object

- 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

The `cin` Object

- User input goes from keyboard to the input buffer, where it is stored as characters
- **`cin`** converts the data to the type that matches the variable

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

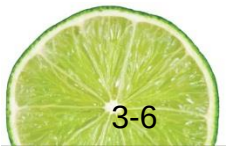


The `cin` Object

- Can be used to input multiple values

```
cin >> height >> width;
```

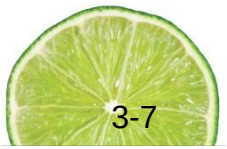
- Multiple values from keyboard must be separated by spaces or [Enter]
- Must press [Enter] after typing last value
- Multiple values need not all be of the same type
- Order is important; first value entered is stored in first variable, etc.



3.2 Mathematical Expressions

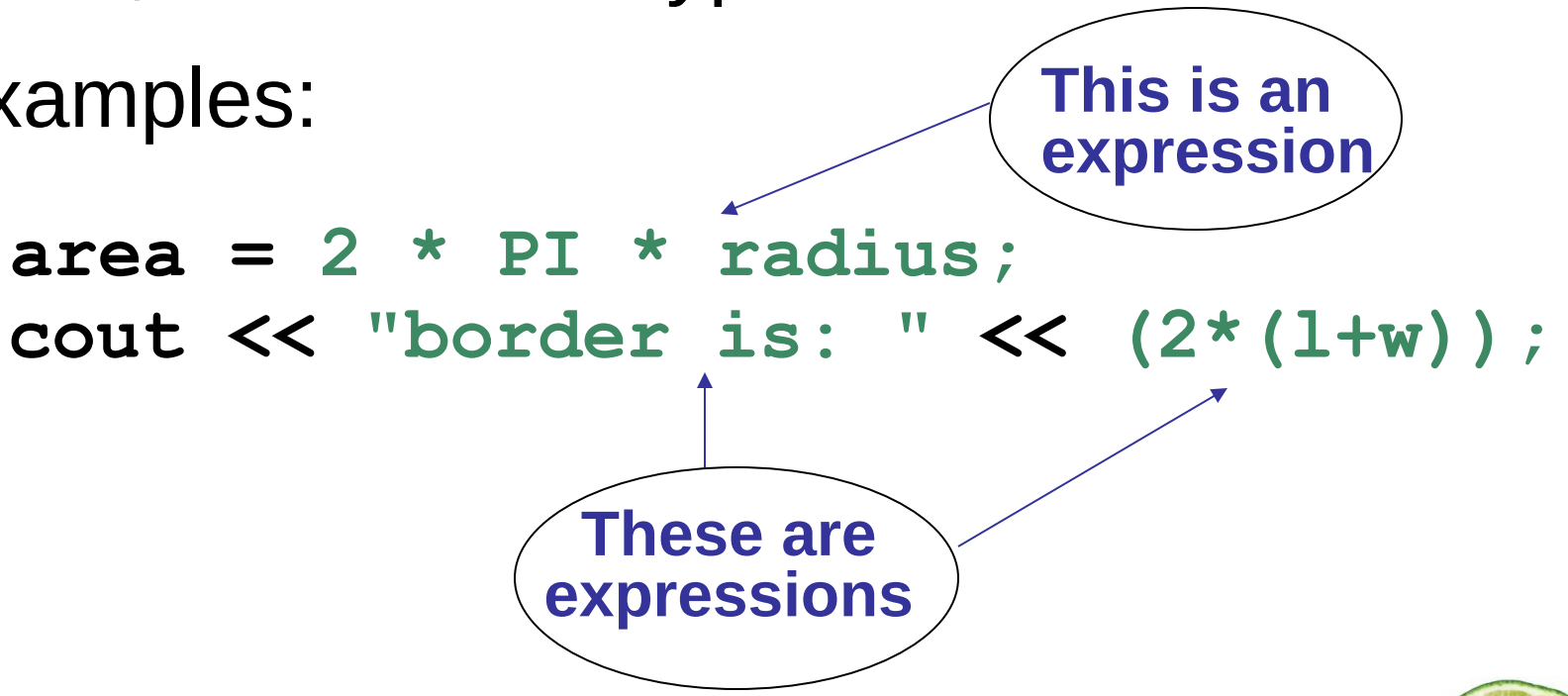
- An expression can be a constant, a variable, or a combination of constants and variables combined with operators
- Can create complex expressions using multiple mathematical operators
- Examples of mathematical expressions:

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

Using Mathematical Expressions

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



Order of Operations

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

Do first: ( ) expressions in parentheses

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

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

Do last: + - in order, left to right

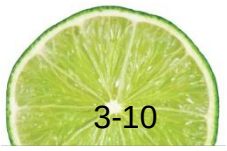
- In the expression `2 + 2 * 2 - 2 ,`

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

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
```



Algebraic Expressions

- Multiplication requires an operator

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

- There is no exponentiation operator

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

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

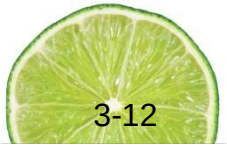
- Parentheses may be needed to maintain order of operations

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

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

3.3 Data Type Conversion and Type Casting

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



Hierarchy of Data Types

- Highest
  - `long double`
  - `double`
  - `float`
  - `unsigned long`
  - `long`
  - `unsigned int`
- Lowest
  - `int`

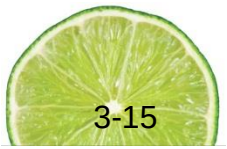
- Ranked by largest number they can hold

Type Coercion

- Coercion: automatic conversion of an operand to another data type
- Promotion: converts to a higher type
- Demotion: converts to a lower type

Coercion Rules

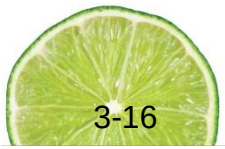
1) **`char`**, **`short`**, **`unsigned short`** are automatically promoted to **`int`**
2) When operating on values of different data types, the lower-ranked one is promoted to the type of the higher one.
3) When using the **`=`** operator, the type of expression on right will be converted to the type of variable on left



Coercion Rules – Important Notes

1) If demotion is required to use the = operator,
   - the stored result may be incorrect if there is not enough space available in the receiving variable
   - floating-point values are truncated when assigned to integer variables

2) Coercion affects the value used in a calculation. It does not change the type associated with a variable.



Type Casting

- Used for manual data type conversion
- Format

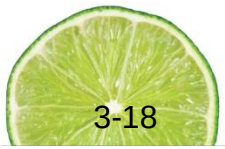
```
static_cast<Data Type>(Value)
```

- Example:

```
cout << static_cast<int>(4.2);
                        // Displays 4
```

More Type Casting Examples

```
char ch = 'C';
cout << ch << " is stored as "
     << static_cast<int>(ch);

gallons = static_cast<int>(area/500);

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



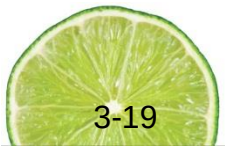
Older Type Cast Styles

```
double Volume = 21.58;
int intVol1, intVol2;
intVol1 = (int) Volume; // C-style
                        // cast
intVol2 = int (Volume); //Prestandard
                        // C++ style
                        // cast
```

C-style cast uses prefix notation

Prestandard C++ cast uses functional notation

`static_cast` is the current standard

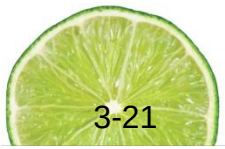


3.4 Overflow and Underflow

- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- The variable contains a value that is ‘wrapped around’ the set of possible values

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
```



Handling Overflow and Underflow

Different systems handle the problem differently. They may

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

3.5 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

Benefits of Named Constants

- Makes program code more readable by documenting the purpose of the constant in the name:

```
const double TAX_RATE = 0.0675;

…

salesTax = purchasePrice * TAX_RATE;
```

- Simplifies program maintenance:

```
const double TAX_RATE = 0.0725;
```

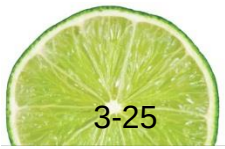
`const` vs. `#define`

`#define`

- C-style of naming constants

  ```
  #define NUM_STATES 50
  ```

- 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`



3.6 Multiple and Combined Assignment

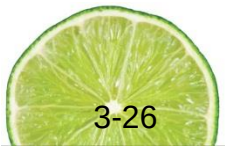
- The assignment operator (=) can be used multiple times in an expression

```
x = y = z = 5;
```

- Associates right to left

```
x = (y = (z = 5));
```

Done 3rd — Done 2nd — Done 1st



Combined Assignment

- Applies an arithmetic operation to a variable and assigns the result as the new value of that variable
- Operators: `+=` `-=` `*=` `/=` `%=`
- Also called compound operators or arithmetic assignment operators
- Example:

```
sum += amt;
```
is short for
```
sum = sum + amt;
```

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);
```

3.7 Formatting Output

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

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. It will use more spaces if specified field width is not big enough.

Stream Manipulators

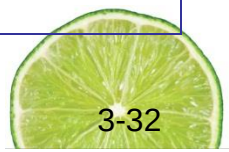
- Some affect values until changed again
  - **`fixed`**: Use decimal notation (not E-notation) for floating-point values.
  - **`setprecision(x)`**:
    - When used with **`fixed`**, print floating-point value using **`x`** digits after the decimal.
    - Without **`fixed`**, print floating-point value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



3.8 Working with Characters and Strings

- **`char`**: holds a single character
- **`string`**: holds a sequence of characters
- Both can be used in assignment statements
- Both can be displayed with **`cout`** and **`<<`**

String Input

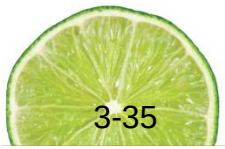
Reading in a string object

```
string str;

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

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

Character Input

Reading in a character:

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

String Operators

= Assigns a value to a string

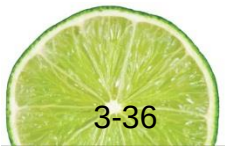
```
string words;
words = "Tasty ";
```

+ 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"
```

string Member Functions

- **`length()`** – the number of characters in a string

```
string firstPrez="George Washington";
int size=firstPrez.length(); // size is 17
```

- assign() – put repeated characters in a string. Can be used for formatting output.

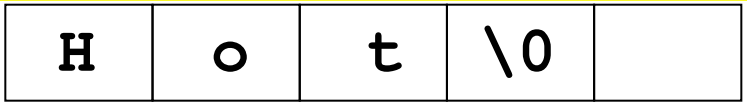
```
string equals;
equals.assign(80,'=');
…
cout << equals << endl;
cout << "Total: " << total << endl;
```

3.9 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`.

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

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
```

C-String and Keyboard Input

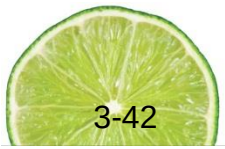
- Must use **`cin.getline()`** to put keyboard input into a C-string
- Note that **`cin.getline() ≠ getline()`**
- Must indicate the target C-string and maximum number of characters to read:

```
const int SIZE = 25;
char name[SIZE];
cout << "What's your name? ";
cin.getline(name, SIZE);
```

3.10 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
`pow`	Raise to a power



More Mathematical Library Functions

These require **`cstdlib`** header file

- **`rand`**
  - Returns a random number between **`0`** and the largest **`int`** the computer holds
  - Will yield the same sequence of numbers each time the program is run
- **`srand(x)`**
  - Initializes random number generator with **`unsigned int x`**. **`x`** is the "seed value".
  - Should be called at most once in a program

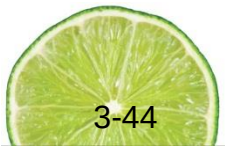
More on Random Numbers

- Use **`time()`** to generate different seed values each time that a program runs:

```
#include <ctime> //needed for time()
…
unsigned seed = time(0);
srand(seed);
```

- Random numbers can be scaled to a range:

```
int max=6;
int num;
num = rand() % max + 1;
```



Chapter 3: Expressions and Interactivity

The End!!!