

THIRD EDITION

A First Book of C++

From Here to There

CHAPTER 3

Assignment and Interactive Input

Objectives

You should be able to describe:

- Assignment Operators
- Mathematical Library Functions
- Interactive Keyboard Input
- Symbolic Constraints
- Common Programming Errors

Assignment Operators

- Basic Assignment Operator:
 - **Format:** *variable = expression;*
 - Computes value of expression on right of = sign, assigns it to variable on left side of = sign
- If not initialized in a declaration statement, a variable should be assigned a value before used in any computation
- Variables can only store one value at a time
 - Subsequent assignment statements will overwrite previously assigned values

Assignment Operators (continued)

- Operand to right of = sign can be:
 - A constant
 - A variable
 - A valid C++ expression
- Operand to left of = sign must be a variable
- If operand on right side is an expression:
 - All variables in expression must have a value to get a valid result from the assignment

Assignment Operators (continued)

- **Expression:** combination of constants and variables that can be evaluated

- Examples

```
Sum = 3 + 7;
```

```
Diff = 15 - 6;
```

```
Product = .05 * 14.6;
```

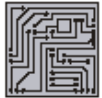
```
Tally = count + 1;
```

```
newtotal = 18.3 + total;
```

```
Average = sum / items;
```

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

Assignment Operators (continued)



Program 3.1

```
// this program calculates the area of a rectangle
//    given its length and width

#include <iostream>
using namespace std;

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

    length = 27.2;
    width = 13.6;
    area = length * width;
    cout << "The length of the rectangle is " << length << endl;
    cout << "The width of the rectangle is " << width << endl;
    cout << "The area of the rectangle is " << area << endl;

    return 0;
}
```

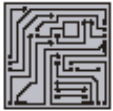
Coercion

- Value on right side of a C++ expression is converted to data type of variable on the left side
- Example:
 - If `temp` is an integer variable, the assignment
`temp = 25.89;`
causes integer value 25 to be stored in integer variable `temp`

Assignment Variations

- `sum = sum + 10;` is a valid C++ expression
 - The value of `sum + 10` is stored in variable `sum`
 - Not a valid algebra equation
- `lvalue`: any valid quantity on left side of assignment operator
- `rvalue`: any valid quantity on right side of assignment operator
- A number can only be an `rvalue`
 - A variable can appear on either side of expression

Assignment Variations (continued)



Program 3.2

```
#include <iostream>
using namespace std;

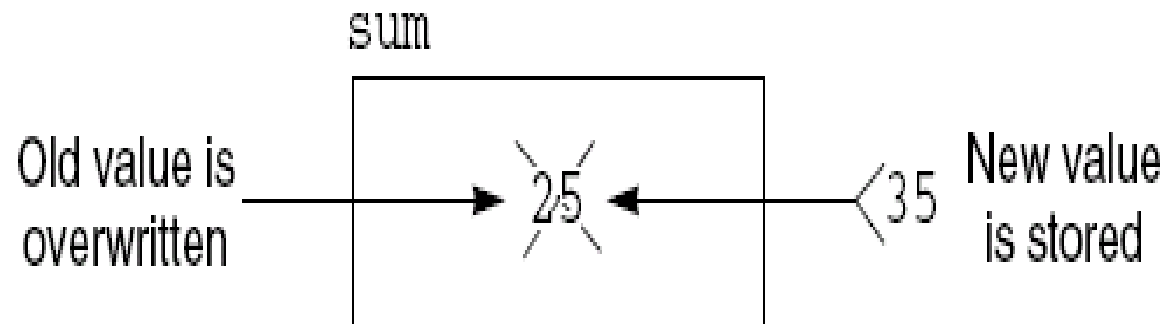
int main()
{
    int sum;

    sum = 25;
    cout << "The number stored in sum is " << sum << endl;
    sum = sum + 10;
    cout << "The number now stored in sum is " << sum << endl;

    return 0;
}
```

Assignment Variations (continued)

FIGURE 3.2 `sum = sum + 10;` Causes a New Value to Be Stored in `sum`.



Assignment Variations (continued)

- Assignment expressions such as:

```
sum = sum + 25;
```

can be by using following shortcut operators:

`+=` `-=` `*=` `/=` `%=`

- Example:

```
sum = sum + 10;
```

can be written as

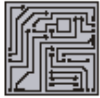
```
sum += 10;
```

Accumulating

- The following statements add the numbers 96, 70, 85 and 60 in calculator fashion:

<u>Statement</u>	<u>Value in <code>sum</code></u>
<code>sum = 0;</code>	0
<code>sum = sum + 96;</code>	96
<code>sum = sum + 70;</code>	166
<code>sum = sum + 85;</code>	251
<code>sum = sum + 60;</code>	311

Accumulating (continued)



Program 3.3

```
#include <iostream>
using namespace std;

int main()
{
    int sum;

    sum = 0;
    cout << "The value of sum is initially set to " << sum << endl;
    sum = sum + 96;
    cout << " sum is now " << sum << endl;
    sum = sum + 70;
    cout << " sum is now " << sum << endl;
    sum = sum + 85;
    cout << " sum is now " << sum << endl;
    sum = sum + 60;
    cout << " The final sum is " << sum << endl;

    return 0;
}
```

Counting

- **Has the form:**

variable = variable + fixedNumber;

- Each time statement is executed, value of variable is increased by a fixed amount

- **Increment Operators (++) , (--)**

- Unary operator for special case when variable is increased or decreased by 1

- Using the increment operator, the expression

variable = variable + 1; can be replaced by
either *++variable;* **or** *variable++;*

Counting (continued)

- Examples of counting statements:

```
i = i + 1;
```

```
n = n + 1;
```

```
count = count + 1;
```

```
j = j + 2;
```

```
m = m + 2;
```

```
kk = kk + 3;
```

Counting (continued)

- Examples of the increment operator:

Expression

`i = i + 1`

`n = n + 1`

`count = count + 1`

Alternative

`i++` or `++i`

`n++` or `++n`

`count++` or `++ count`

Counting (continued)



Program 3.4

```
#include <iostream>
using namespace std;

int main()
{
    int count;

    count = 0;
    cout << "The initial value of count is " << count << endl;
    count++;
    cout << " count is now " << count << endl;
    count++;
    cout << " count is now " << count << endl;
    count++;
    cout << " count is now " << count << endl;
    count++;
    cout << " count is now " << count << endl;

    return 0;
}
```

Counting (continued)

The output displayed by Program 3.4 is:

```
The initial value of count is 0  
count is now 1  
count is now 2  
count is now 3  
count is now 4
```

Counting (continued)

- **Prefix increment operator:** the ++ or -- operator appears before a variable

- The expression `k = ++n` does two things

```
n = n + 1; // increment n first
```

```
k = n; // assign n's value to k
```

- **Postfix increment operator:** the ++ or -- operator appears after a variable

- The expression `k = n++` works differently

```
k = n; // assign n's value to k
```

```
n = n + 1; // and then increment n
```

Mathematical Library Functions

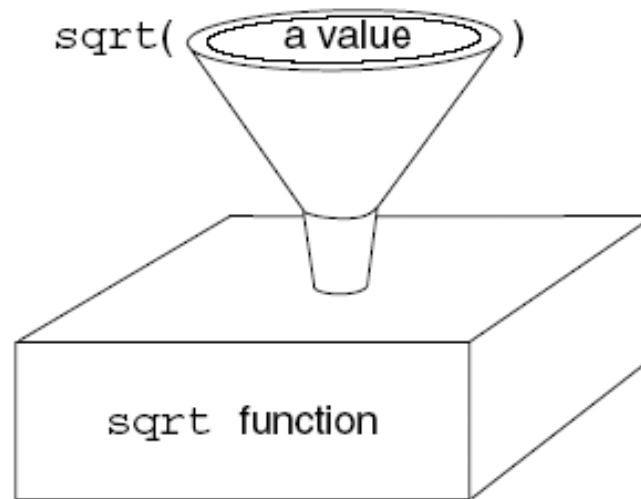
- Standard preprogrammed functions that can be included in a program
 - **Example:** `sqrt(number)` calculates the square root of `number`
- Table 3.1 lists more commonly used mathematical functions provided in C++
 - To access these functions in a program, the header file `cmath` must be used
 - **Format:** `#include <cmath>` <- no semicolon

Mathematical Library Functions (continued)

- Before using a C++ mathematical function the programmer must know:
 - Name of the desired mathematical function
 - What the function does
 - Type of data required by the function
 - Data type of the result returned by the function

Mathematical Library Functions (continued)

FIGURE 3.3 *Passing Data to the `sqrt()` Function*



Mathematical Library Functions (continued)

TABLE 3.1 *Common C++ Functions*

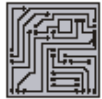
Function Name	Description	Returned Value
<code>abs(x)</code>	absolute value	same data type as argument
<code>pow(x1, x2)</code>	x_1 raised to the x_2 power	data type of argument x_1
<code>sqrt(x)</code>	square root of x	double
<code>sin(x)</code>	sine of x (x in radians)	double
<code>cos(x)</code>	cosine of x (x in radians)	double
<code>tan(x)</code>	tangent of x (x in radians)	double
<code>log(x)</code>	natural logarithm of x	double
<code>log10(x)</code>	common log (base 10) of x	double
<code>exp(x)</code>	e raised to the x power	double

Mathematical Library Functions (continued)

TABLE 3.2 *Selected Examples of Functions*

Example	Returned Value
<code>abs (-7.362)</code>	7.362
<code>abs (-3)</code>	3
<code>pow (2.0, 5.0)</code>	32
<code>pow (10, 3)</code>	1000
<code>log (18.697)</code>	2.92836
<code>log10 (18.697)</code>	1.27177
<code>exp (-3.2)</code>	

Mathematical Library Functions (continued)



Program 3.5

```
#include <iostream> // this line may be placed second instead of first
#include <cmath>     // this line may be placed first instead of second
using namespace std;

int main()
{
    int height;
    double time;

    height = 800;
    time = sqrt(2 * height / 32.2);
    cout << "It will take " << time << " seconds "
         << "to fall " << height << " feet.\n";
    return 0;
}
```

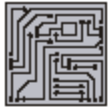
Casts

- **Cast:** forces conversion of a value to another type
 - **Two versions:** compile time and run time
- **Compile-time cast:** unary operator with syntax
`dataType (expression)`
 - `expression` converted to data type of `dataType`
- **Run-time cast:** requested conversion checked at runtime, applied if valid
 - **Syntax:** `staticCast<dataType>(expression)`
 - `expression` converted to data type `dataType`

Interactive Keyboard Input

- If a program only executes once, data can be included directly in the program
 - If data changes, program must be rewritten
 - Capability needed to enter different data
- **cin object:** used to enter data while a program is executing
 - **Example:** `cin >> num1;`
 - Statement stops program execution and accepts data from the keyboard

Interactive Keyboard Input (continued)



Program 3.6

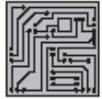
```
#include <iostream>
using namespace std;

int main()
{
    double num1, num2, product;

    num1 = 30.0;
    num2 = 0.05;
    product = num1 * num2;
    cout << "30.0 times 0.05 is " << product << endl;

    return 0;
}
```

Interactive Keyboard Input (continued)



Program 3.8

```
#include <iostream>
using namespace std;

int main()
{
    double num1, num2, product;

    cout << "Please type in a number: ";
    cin  >> num1;
    cout << "Please type in another number: ";
    cin  >> num2;
    product = num1 * num2;
    cout << num1 << " times " << num2 << " is " << product << endl;

    return 0;
}
```

Interactive Keyboard Input (continued)

- First `cout` statement in Program 3.8 prints a string
 - Tells the person at the terminal what to type
 - A string used in this manner is called a **prompt**
- Next statement, `cin`, pauses computer
 - Waits for user to type a value
 - User signals the end of data entry by pressing Enter key
 - Entered value stored in variable to right of extraction symbol
- Computer comes out of pause and goes to next `cout` statement

A First Look at User-Input Validation

- A well-constructed program should validate all user input
 - Ensures that program does not crash or produce nonsensical output
- **Robust Programs:** programs that detect and respond effectively to unexpected user input
 - Also known as *bullet-proof* programs
- **User-input validation:** validating entered data and providing user with a way to re-enter invalid data

Symbolic Constants

- **Magic Numbers:** literal data used in a program
 - Some have general meaning in context of program
tax rate in a program to calculate taxes
 - Others have general meaning beyond the context of the program

$\pi = 3.1416$, Euler's number = 2.71828

- Constants can be assigned symbolic names

```
const float PI = 3.1416f;
```

```
const double SALESTAX = 0.05;
```

Symbolic Constants (continued)

- **const**: qualifier specifies that the declared identifier cannot be changed
- A `const` identifier can be used in any C++ statement in place of number it represents

```
circum = 2 * PI * radius;
amount = SALESTAX * purchase;
```
- `const` identifiers commonly referred to as:
 - symbolic constants
 - named constants

Placement of Statements

- A variable or symbolic constant must be declared before it is used
- C++ permits preprocessor directives and declaration statements to be placed anywhere in program
 - Doing so results in very poor program structure

Placement of Statements (continued)

- As a matter of good programming practice, the order of statements should be:

```
preprocessor directives
int main()
{
    symbolic constants
    variable declarations
    other executable statements
    return value
}
```

Placement of Statements (continued)



Program 3.10

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    const double SALESTAX = 0.05;
    double amount, taxes, total;

    cout << "\nEnter the amount purchased: ";
    cin >> amount;
    taxes = SALESTAX * amount;
    total = amount + taxes;
    cout << setiosflags(ios::fixed)
         << setiosflags(ios::showpoint)
         << setprecision(2);
    cout << "The sales tax is " << setw(4) << taxes << endl;
    cout << "The total bill is " << setw(5) << total << endl;

    return 0;
}
```

Common Programming Errors

- Forgetting to assign or initialize values for all variables before they are used in an expression
- Applying increment or decrement operator to an expression

`(count + n)++` is incorrect

- Increment and decrement operators can only be applied to individual variables

Common Programming Errors (continued)

- Forgetting to separate all variables passed to `cin` with an extraction symbol, `>>`
- Using an increment or decrement operator with variables that appear more than once in the same statement

Summary

- Expression: sequence of operands separated by operators
- Expressions are evaluated according to precedence and associativity of its operands
- The assignment symbol, =, is an operator
 - Assigns a value to variable
 - Multiple assignments allowed in one statement
- Increment operator(++): adds 1 to a variable
- Decrement operator(--): subtracts 1 from a variable

Summary (continued)

- Increment and decrement operators can be used as prefixes or postfixes
- C++ provides library functions for various mathematical functions
 - These functions operate on their arguments to calculate a single value
 - Arguments, separated by commas, included within parentheses following function's name
- Functions may be included within larger expressions

Summary (continued)

- `cin` object used for data input
- `cin` temporarily suspends statement execution until data entered for variables in `cin` function
- **Good programming practice:** prior to a `cin` statement, display message alerting user to type and number of data items to be entered
 - Message called a **prompt**
- Values can be equated to a single constant by using the `const` keyword