COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 3

Numeric Types, Expressions, and Output

Chapter 3 Topics

- Chapter 3 Examples
- Constants of Type int and float
- Evaluating Arithmetic Expressions
- Implicit Type Coercion and Explicit Type Conversion
- Calling a Value-Returning Function
- Using Function Arguments

Chapter 3 Topics

- **Using C++ Library Functions in Expressions**
- **Calling a Void Function**

examples

- download Book Code/Chapter03 as a zip file.

Samples of C++ Data Values

int sample values

4578 -4578 0

float sample values

95.274 95. .265

9521E-3 -95E-1 95.213E2

char sample values

'B' 'd' '4' '?' '*'

Scientific Notation

2.7E4 means $2.7 \times 10^{4} =$

$2.7000 =$

27000.0

2.7E-4 means $2.7 \times 10^{-4} =$

$0002.7 =$

0.00027

More About Floating Point Values

- **Floating point numbers** have an **integer part** and a **fractional part**, with a decimal point in between.
- Either the integer part or the fractional part, but not both, may be missing

Examples 18.4 500. .8

- 127.358

More About Floating Point Values

- Alternatively, floating point values can have an **exponent**, as in scientific notation
- The number preceding the letter E doesn't need to include a decimal point

Examples 1.84E1 5E2 8E-1

-.127358E3

Division Operator

- The result of the division operator depends on the type of its operands
- If one or both operands has a floating point type, the result is a floating point type.
- Otherwise, the result is an integer type

- Examples

11 / 4	has value	2
11.0 / 4.0	has value	2.75
11 / 4.0	has value	2.75

Main returns an int value to the operating system

```
//*****************************************************************
// FreezeBoil program
// This program computes the midpoint between
// the freezing and boiling points of water
//*****************************************************************
#include <iostream>
using namespace std;
const float FREEZE_PT = 32.0; // Freezing point of water
const float BOIL_PT = 212.0; // Boiling point of water

int main() {
  float avgTemp; // Holds the result of averaging
                 //  FREEZE_PT and BOIL_PT

  cout << "Water freezes at " << FREEZE_PT << endl;
  cout << " and boils at " << BOIL_PT << " degrees." << endl;
  avgTemp = FREEZE_PT + BOIL_PT;
  avgTemp = avgTemp / 2.0;
  cout << "Halfway between is ";
  cout << avgTemp << " degrees." << endl;
  return 0;
}
```

Modulus Operator

- The **modulus operator** % can only be used with integer type operands and **always has an integer type result**
- Its result is the integer type **remainder** of an integer division
- Example

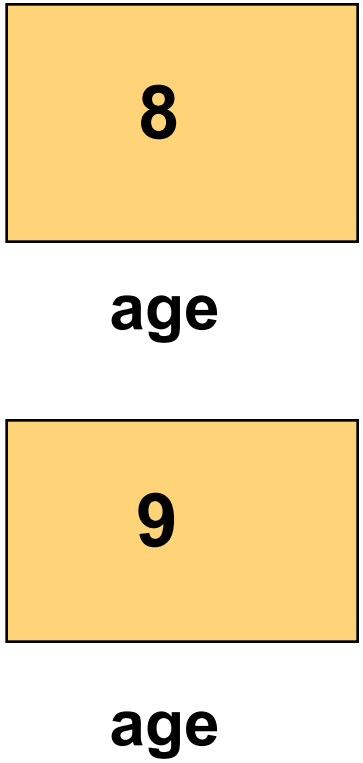
11 % 4 has value 3 because

$$R = ?$$

$$4 \overline{)\ 11}$$

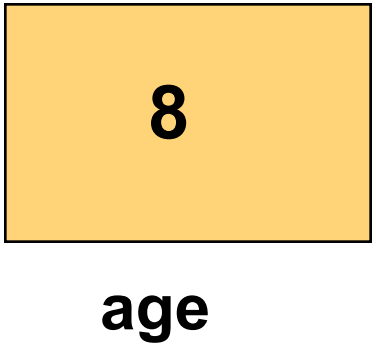
More C++ Operators

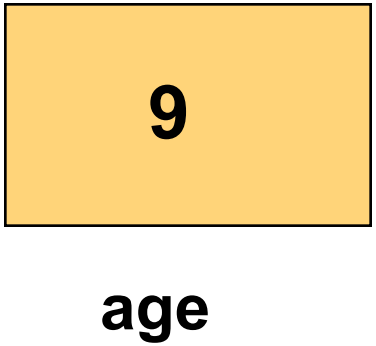
```
int  age;

age = 8;

age = age + 1;
```



Prefix Form Increment Operator

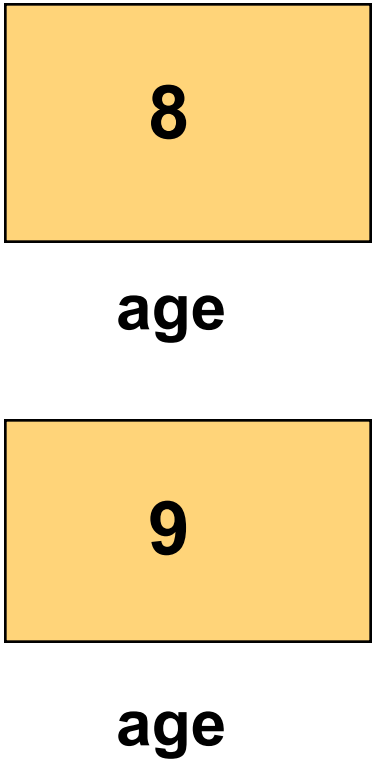
```
int  age;

age = 8;

++age;
```





Postfix Form Increment Operator

```
int  age;

age = 8;

age++;
```



Decrement Operator

```
int  dogs;

dogs = 100;

dogs--;
```

Which Form to Use

- When the increment(or decrement) operator is used in a "*stand alone*" statement solely to add one(or subtract one) from a variable's value, it can be used in either prefix or postfix form

dogs--;

--dogs;

BUT...

- **When the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield *different* results**

We'll see how later . . .

What is an Expression in C++?

- An **expression** is a valid arrangement of variables, constants, and operators
- In C++ each expression can be evaluated to compute a value of a given type
- The value of the expression

  9.3 * 4.5 is 41.85

Operators can be

binary	**involving 2 operands**	**2 + 3**
unary	**involving 1 operand**	**- 3**
ternary	**involving 3 operands**	***later***

Some C++ Operators

Precedence	Operator	Description
Higher	( )	Function call
	+	Positive
	-	Negative
	*	Multiplication
	/	Division
	%	Modulus(remainder)
	+	Addition
	-	Subtraction
Lower	=	Assignment

Precedence

- Higher Precedence determines which operator is applied first in an expression having several operators

Associativity

- Left to right **associativity**–in an expression having two operators with the same priority, the left operator is applied first

- **Grouping order** –synonmous w/ associativity

- In C++ the binary operators

  *, /, %, +, - are all left associative

- Expression 9 - 5 - 1 means (9 - 5) - 1

  4 - 1

  3

Evaluate the Expression

7 * 10 - 5 % 3 * 4 + 9

(7 * 10) - 5 % 3 * 4 + 9

70 - 5 % 3 * 4 + 9

70 -(5 % 3) * 4 + 9

70 - 2 * 4 + 9

70 -(2 * 4) + 9

70 - 8 + 9

(70 - 8) + 9

62 + 9

71

Parentheses

- Parentheses can be used to change the usual order
- Parts in() are evaluated first
- Evaluate (7 *(10 - 5) % 3) * 4 + 9

```
(7 *(10 - 5) % 3) * 4 + 9
(7 * 5 % 3 ) * 4 + 9
( 35 % 3) * 4 + 9
2 * 4 + 9
8 + 9
17
```

Recall Assignment Operator Syntax

Variable = Expression

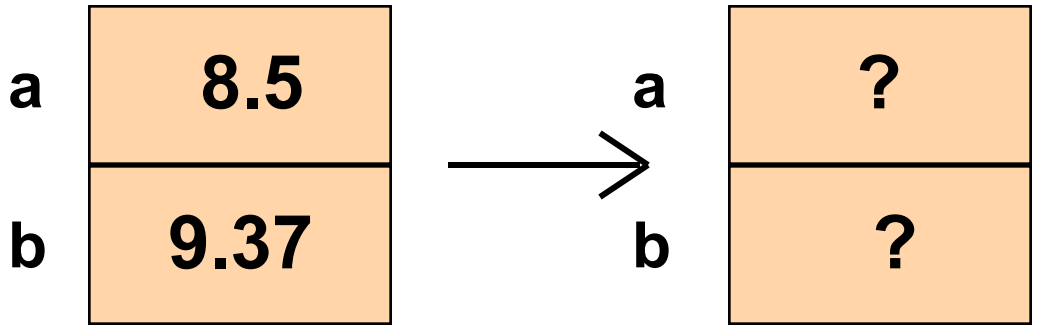
- First, expression on right is evaluated
- Then the resulting value is stored in the memory location of variable on left

Automatic Type Conversion

- Implict conversion by the compiler of a value from one data type to another is known as automatic type coercion
- An automatic type coercion occurs after evaluation but before the value is stored if the types differ for expression and variable
- See examples on Slides 31, 32, and 33

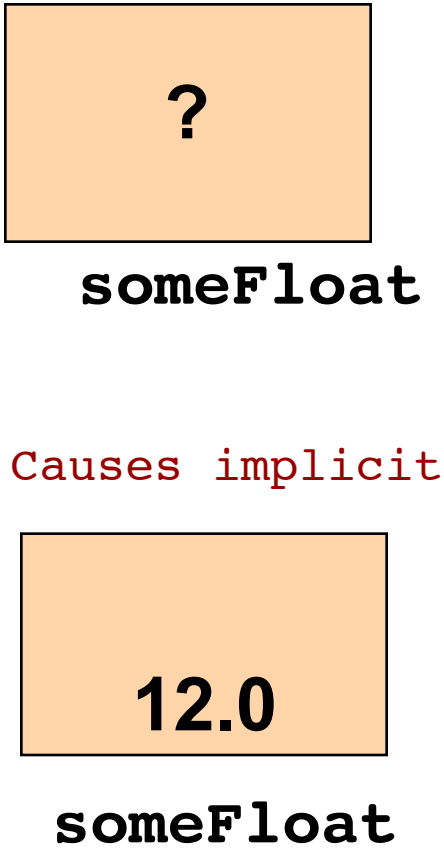
What value is stored?

```
float  a;
float  b;

a = 8.5;
b = 9.37;
a = b;
```



What is stored?

```
float someFloat;

someFloat = 12;   // Causes implicit type conversion
```



What is stored?

```
int  someInt;


someInt = 4.8;
```

Type Casting is Explicit Conversion of Type

- **Explicit type casting** (or **type conversion**) used to clarify that the mixing of types is intentional, not an oversight
- Explicit type casting helps make programs clear and error free as possible

Examples of Explicit Typecasting

int(4.8)	has value	4
float(5)	has value	5.0
float(7/4)	has value	1.0
float(7) / float(4)	has value	1.75

Some Expressions

```
int  age;
```

Example	Value
age = 8	8
- age	- 8
5 + 8	13
5 / 8	0
6.0 / 5.0	1.2
float(4 / 8)	0.0
float(4) / 8	0.5
cout << “How old are you?”	cout
cin >> age	cin
cout << age	cout

What values are stored?

```
float  loCost;
float  hiCost;

loCost = 12.342;
hiCost = 12.348;

loCost =
   float(int(loCost * 100.0 + 0.5)) / 100.0;

hiCost  =
   float(int(hiCost * 100.0 + 0.5)) / 100.0;
```

Values were rounded to 2 decimal places

12.34

loCost

12.35

hiCost

Functions

- Every C++ program must have a function called `main`
- Program execution always begins with function `main`
- Any other functions are subprograms and must be called by the `main` function

Function Calls

- One function calls another by using the name of the called function together with() containing an argument list
- A function call temporarily transfers control from the calling function to the called function

More About Functions

- It is not considered good practice for the body block of function main to be long
- Function calls are used to do subtasks
- Every C++ function has a return type
- If the return type is not void, the function returns a value to the calling block

Where are functions?

Functions are subprograms

- **located in libraries, or**
- **written by programmers for their use in a particular program**

HEADER FILE	FUNCTION	EXAMPLE OF CALL	VALUE
<cstdlib>	abs(i)	abs(-6)	6
<cmath>	pow(x,y)	pow(2.0,3.0)	8.0
	fabs(x)	fabs(-6.4)	6.4
<cmath>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<cmath>	log(x)	log(2.0)	.693147
<iomanip>	setprecision(n)	setprecision(3)	

Write C++ Expressions for

The square root of $b^2 - 4ac$

```
sqrt(b * b - 4.0 * a * c)
```

The square root of the average of myAge and yourAge

```
sqrt((myAge  +  yourAge) / 2)
```

Function Call

- A **function call** temporarily **transfers control** to the called function's code
- When the function's code has finished executing, **control is transferred back** to the calling block

Function Call Syntax

Function Name = (Argument List)

- **The argument list is a way for functions to communicate with each other by passing information**

- **The argument list can contain zero, one, or more arguments, separated by commas, depending on the function**

A void function call stands alone

```
#include <iostream>

void  DisplayMessage(int  n);
// Declares function

int main()
{
    DisplayMessage(15);
    // Function call
    cout  <<  "Good Bye"  <<  endl;
    return 0;
}
```

A void function does NOT return a value

```
// Header and body here

void  DisplayMessage(int  n)
{
    cout  <<  "I have liked math for  "
          <<  n  <<  " years"  << endl;
}
```

Two Kinds of Functions

Value-Returning	Void
Always returns a single value to its caller and is called from within an expression	Never returns a value to its caller and is called as a separate statement

<< is a binary operator

<< is called the output or insertion operator

<< is left associative

Expression	Has value
cout << age	cout

Statement

```
cout << "You are "  << age  <<  " years old\n";
```

<iostream> is header file

- For a library that defines 3 objects

  An istream object named cin (keyboard)

  An ostream object named cout (screen)

  An ostream object named cerr (screen)

No I/O is built into C++

- Instead, a library provides input stream and output stream