Chapter 2: Elementary Programming

Sections 2.1–2.13, 2.15, 2.16

Textbooks: Y. Daniel Liang, Introduction to Programming with C++, 3rd Edition

These slides were adapted by Prof. Gheith Abandah from the Computer Engineering Department of the University of Jordan for the Course: Computer Skills for Engineers (0907101)
Updated by Dr. Ashraf Suyyagh (Spring 2021)

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Writing a Simple Program

A program that computes the area of the circle.

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

ComputeArea

Run

Note: Clicking the blue button runs the code from Windows. If you cannot run the buttons, see

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc.

animation

Trace the Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << endl;
}
```

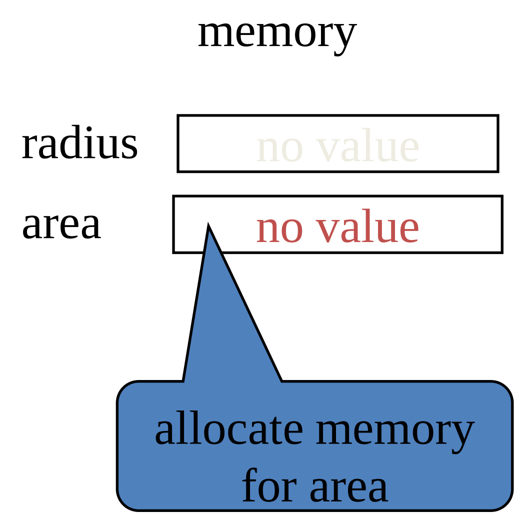
allocate memory for radius

radius | no value

animation

Trace the Program Execution

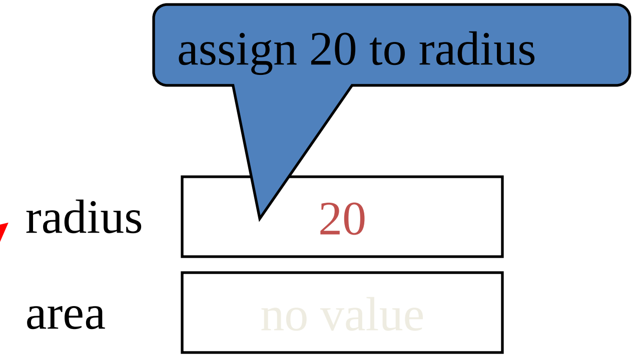
```
#include <iostream>
using namespace std;

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

Trace the Program Execution

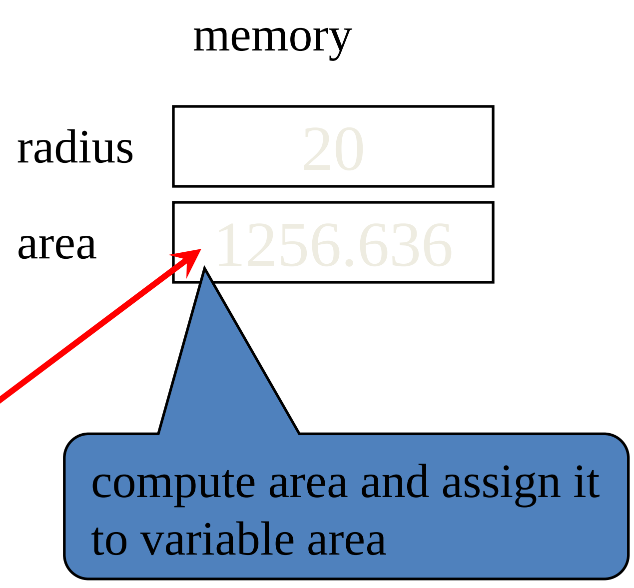
```
#include <iostream>
using namespace std;

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

Trace the Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

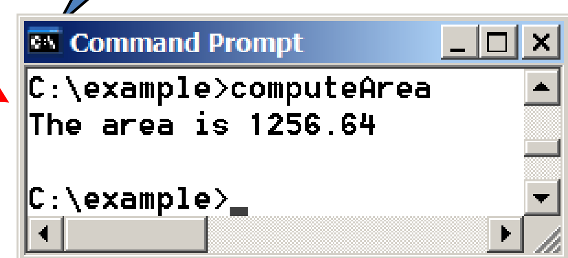
Trace the Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```

memory

radius	20
area	1256.636

print a message to the console



Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Reading Input from the Keyboard

You can use the **`cin`** object to read input from the keyboard.

```
cin >> radius;
```

ComputeAreaWithConsoleInput Run

Reading Multiple Input in One Statement

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

int main()
{
  // Prompt the user to enter three numbers
  double number1, number2, number3;
  cout << "Enter three numbers: ";
  cin >> number1 >> number2 >> number3;

  // Compute average
  double average = (number1 + number2 + number3) / 3;

  // Display result
  cout << "The average of " << number1 << " " << number2
      << " " << number3 << " is " << average << endl;

  return 0;
}
```

ComputeAverage

Run

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Identifiers

Identifiers are the names that identify elements such as variables and functions in a program.

- An identifier is a sequence of characters that consists of letters, digits, and underscores (_).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "C++ Keywords," for a list of reserved words.)
- An identifier can be of any length, but your C++ compiler may impose some restriction. Use identifiers of 31 characters or fewer to ensure portability.

Which of the following identifiers are valid? Which are C++ keywords?

```
miles, Test, a++, --a, 4#R, $4, #44, apps
main, double, int, x, y, radius
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Variables

Variables are used to represent values that may be changed in the program.

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
cout << area;


// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
cout << area;
```

Declaring Variables

```
datatype variable1, variable2,..., variablen;

int x;             // Declare x to be an
                   // integer variable;
double radius; // Declare radius to
                   // be a double variable;
char a;            // Declare a to be a
                   // character variable;
```

Declaring Variables

```
int i, j, k;    // Declare three integers

int i = 10;     // Declare and initialize

int i(1), j(2); // Is equivalent to
int i = 1, j = 2;
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Assignment Statements

An assignment statement designates a value for a variable. An assignment statement can be used as an expression in C++.

```
x = 1;            // Assign 1 to x;
y = x + 1;        // Assign 2 to y;
radius = 1.0;     // Assign 1.0 to radius;
a = 'A';          // Assign 'A' to a;
```

Assignment Statements

An assignment statement designates a value for a variable.

```
i = j = k = 1; // Assigns 1 to the three
               // variables

cout << x = 1; // Assigns 1 to x and
               // outputs 1
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Named Constants

A named constant is an identifier that represents a permanent value.

```
const datatype CONSTANTNAME = VALUE;
```

```
const double PI = 3.14159;
```

ComputeAreaConstant   Run

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Numerical Data Types

- **Signed integers**
  - 16 bits: `short` -3
  - 32 bits: `int` 100000
  - 64 bits: `long long` -2147483648
- **Unsigned integers**
  - 16 bits: `unsigned short` 4
  - 32 bits: `unsigned`
  - 64 bits: `unsigned long long`

Synonymous Types

`short int` is synonymous to **`short`**. For example,

```
    short int i = 2;
```

is same as

```
    short i = 2;
```

```
unsigned short int    ≡ unsigned short
unsigned int          ≡ unsigned
long int              ≡ long
unsigned long int     ≡ unsigned long
```

Numerical Data Types

- **Floating-point numbers**
  - 32 bits: **`float`** `1.5`
  - 64 bits: **`double`** `-1.23456E+2`
  - 80 bits: **`long double`** `9.1e-1000`

- When a number such as 50.534 is converted into scientific notation such as 5.0534e+1, its decimal point is moved (i.e., floated) to a new position.

`double` vs. `float`

The double type values are more accurate than the float type values. For example,

```
cout << "1.0 / 3.0 is " << 1.0 / 3.0 << endl;
```

```
1.0 / 3.0 is 0.33333333333333331
```

16 digits

```
cout << "1.0F / 3.0F is " << 1.0F / 3.0F << endl
```

```
1.0F / 3.0F is 0.3333333432674408
```

7 digits

Numerical Data Types

Name	Synonymy	Range	Storage Size
short	short int	-2^{15} to $2^{15}-1$ (-32,768 to 32,767)	16-bit signed
unsigned short	unsigned short int	0 to $2^{16}-1$ (65535)	16-bit unsigned
int	signed	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit
unsigned	unsigned int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long	long int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
unsigned long	unsigned long int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long long		-2^{63} (-9223372036854775808) to 263-1 (9223372036854775807)	64-bit signed
float		Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double		Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754
long double		Negative range: -1.18E+4932 to -3.37E-4932 Positive range: 3.37E-4932 to 1.18E+4932 Significant decimal digits: 19	80-bit

`sizeof` Function

You can use the **`sizeof`** function to find the size of a type. For example, the following statement displays the size of **`int`**, **`long`**, and **`double`** on your machine.

```
cout << sizeof(int) << " " <<
  sizeof(long) << " " << sizeof(double);
4 4 8
```

```
double area = 5.4;
cout << "Size of area: " << sizeof(area)
  << " bytes" << endl;
Size of area: 8 bytes
```

Numeric Literals

A *literal* is a constant value that appears directly in a program. For example, 34, 1000000, and 5.0 are literals in the following statements:

```
int i = 34;
long k = 1000000;
double d = 5.0;
```

octal and hex literals

- By default, an integer literal is a *decimal* number.
- To denote a *binary* integer literal, use a leading **`0b`** or **`0B`** (zero b).
- To denote an *octal* integer literal, use a leading **`0`** (zero)
- To denote a *hexadecimal* integer literal, use a leading **`0x`** or **`0X`** (zero x).

```
cout << 10 << " " << 0b10 << " " << 010
  << " " << 0x10;
10 2 8 16
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Numeric Operators

Operator	*Name*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Modulus	20 % 3	2

Integer Division

5 / 3 yields an integer 1.

5.0 / 2 yields a double value 2.5

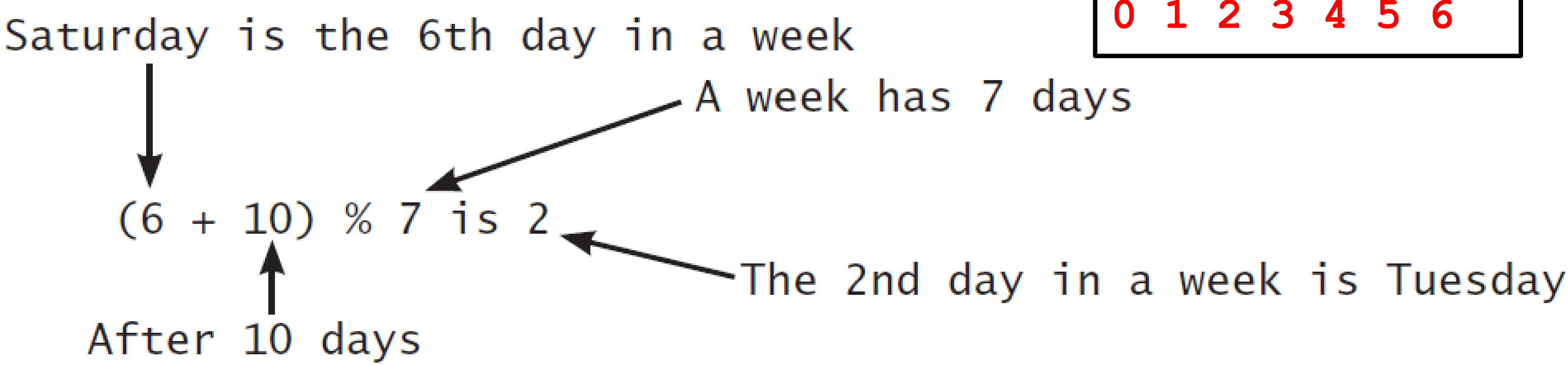
5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

S	M	T	W	T	F	S
0	1	2	3	4	5	6



Example: Displaying Time

A program that obtains minutes from seconds.

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

int main()
{
  // Prompt the user for input
  int seconds;
  cout << "Enter an integer for seconds: ";
  cin >> seconds;
  int minutes = seconds / 60;
  int remainingSeconds = seconds % 60;
  cout << seconds << " seconds is " << minutes <<
    " minutes and " << remainingSeconds << " seconds " << endl;

  return 0;
}
```

DisplayTime

Run

Exponent Operations

`pow(a, b)` $= a^b$

```
cout << pow(2.0, 3) << endl;
8
cout << pow(4.0, 0.5) << endl;
2
cout << pow(2.5, 2) << endl;
6.25
cout << pow(2.5, -2) << endl;
0.16
```

Overflow

When a variable is assigned a value that is too large to be stored, it causes *overflow*.

For example, executing the following statement causes overflow, because the largest value that can be stored in a variable of the **`short`** type is 32767. 32768 is too large.

```
short value = 32767 + 1;
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Arithmetic Expressions

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

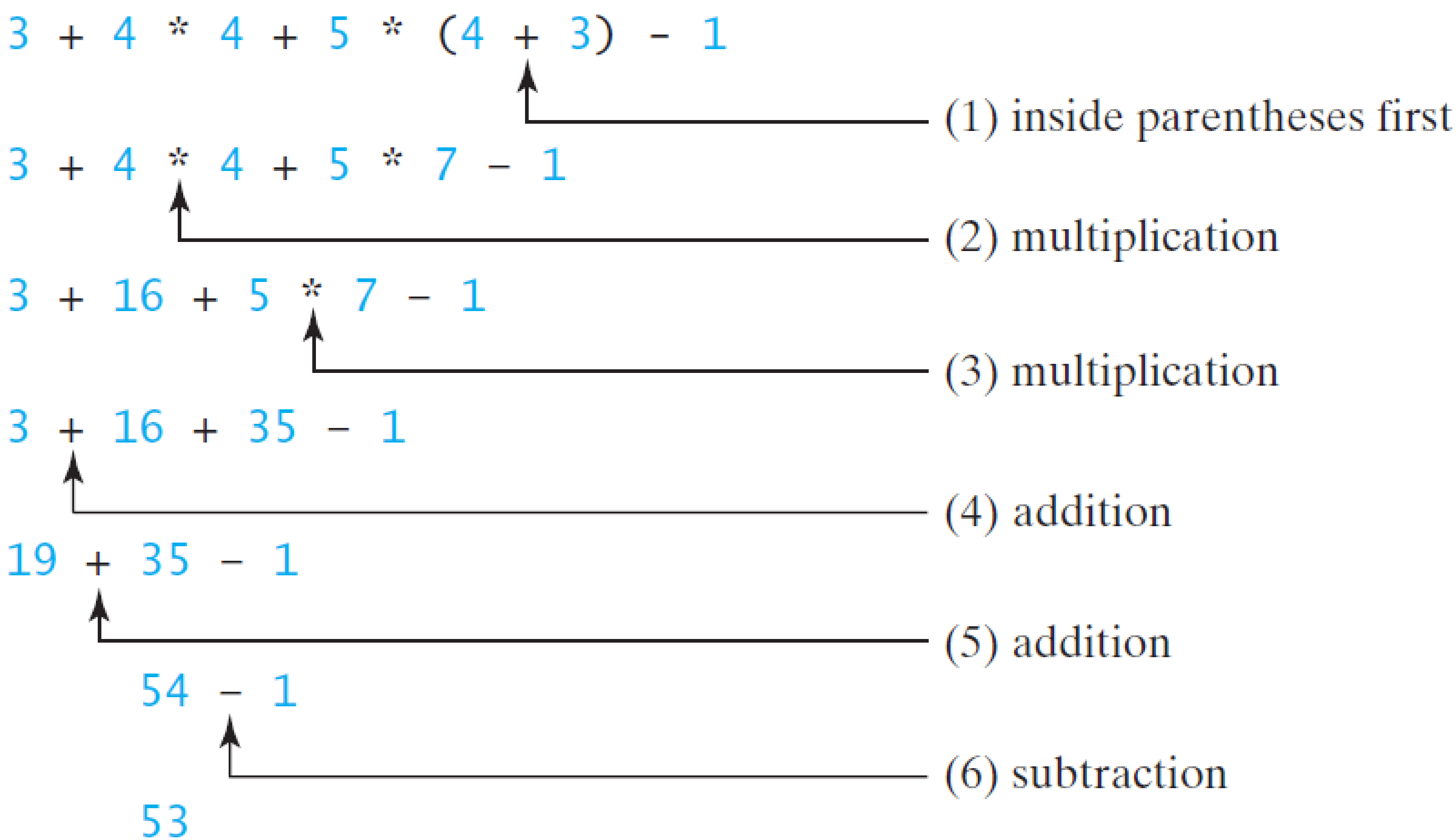
is translated to

```
(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x +
    (9+x)/y)
```

Precedence

`( )`	Operators contained within pairs of parentheses are evaluated first.
`* / %`	Multiplication, division, and remainder operators are applied next.
`+ -`	Addition and subtraction operators are applied last.
→	If an expression contains several similar operators, they are applied from left to right.

Precedence Example



Example: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

```
double celsius = (5.0 / 9) * (fahrenheit - 32);
```

FahrenheitToCelsius

Run

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Displaying the Current Time

Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

The **`time(0)`** function in the **`ctime`** header file returns the current time in seconds elapsed since the time 00:00:00 on January 1, 1970 GMT, as shown in Figure 2.1. This time is known as the Unix epoch because 1970 was the year when the Unix operating system was formally introduced.

ShowCurrentTime Run

ShowCurrentTime.cpp

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
  // Obtain the total seconds since the midnight, Jan 1, 1970
  int totalSeconds = time(0);
  // Compute the current second in the minute in the hour
  int currentSecond = totalSeconds % 60;
  // Obtain the total minutes
  int totalMinutes = totalSeconds / 60;
  // Compute the current minute in the hour
  int currentMinute = totalMinutes % 60;
  // Obtain the total hours
  long totalHours = totalMinutes / 60;
  // Compute the current hour
  int currentHour = (int)(totalHours % 24);
  // Display results
  cout << "Current time is " << currentHour << ":"
    << currentMinute << ":" << currentSecond << " GMT" << endl;
  return 0;
}
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Modulus assignment	`i %= 8`	`i = i % 8`

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Increment and Decrement Operators

Operator	Name	Description
`++var`	pre-increment	Increments **`var`** by 1 and evaluates to the new value in **`var`** after the increment.
`var++`	post-increment	Evaluates to the original value in **`var`** and increments **`var`** by 1.
`--var`	pre-decrement	Decrements **`var`** by 1 and evaluates to the new value in **`var`** after the decrement.
`var--`	post-decrement	Evaluates to the original value in **`var`** and decrements **`var`** by 1.

Increment and Decrement Operators, cont.

What is the output of the following two sequences?

```
int i = 10;
int newNum = 10 * i++;
cout << "i is " << i
  << ", newNum is " << newNum;
```

Same effect as →

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
cout << "i is " << i
  << ", newNum is " << newNum;
```

Same effect as →

```
i = i + 1;
int newNum = 10 * i;
```

Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this:

```
int k = ++i + i;  // Avoid!
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Numeric Type Conversion

Consider the following statements:

```
short i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

```
int i = 34.7;        // i becomes 34
double f = i;        // f is now 34
double g = 34.3;     // g becomes 34.3
int j = g;           // j is now 34
```

Type Casting

Implicit casting

```
double d = 3; // type widening
```

Explicit casting

```
int i = static_cast<int>(3.0);
        // type narrowing
int i = (int)3.9; // C-style casting
        // Fraction part is truncated
```

NOTE

Casting does not change the variable being cast. For example, **d** is not changed after casting in the following code:

```
double d = 4.5;
int i = static_cast<int>(d);
      // d is not changed
```

NOTE

The GNU and Visual C++ compilers will give a warning when you narrow a type unless you use **`static_cast`** to make the conversion explicit.

Example: Keeping Two Digits after Decimal Points

Write a program that displays the 6%-sales tax with two digits after the decimal point.

```
cout << "Sales tax is " <<
   static_cast<int>(tax * 100) / 100.0;
```

SalesTax Run

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Case Study: Counting Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies.

Dollar = 100 cents

Quarters = 25 cents

Dime = 10 cents

Nickel = 5 cents

ComputeChange Run

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining
  amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining
  amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining
  amount
int numberOfPennies = remainingAmount;
```

remainingAmount 1156

remainingAmount initialized

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining
   amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining
   amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining
   amount
int numberOfPennies = remainingAmount;
```

remainingAmount	1156
numberOfOneDollars	11

numberOfOneDollars assigned

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining
  amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining
  amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining
  amount
int numberOfPennies = remainingAmount;
```

remainingAmount 56

numberOfOneDollars 11

remainingAmount updated

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining
  amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining
  amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining
  amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11
numberOfOneQuarters	2

numberOfOneQuarters assigned

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining
  amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining
  amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining
  amount
int numberOfPennies = remainingAmount;
```

remainingAmount	6
numberOfOneDollars	11
numberOfQuarters	2

remainingAmount updated

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors

Common Errors

1. Undeclared or Uninitialized Variables

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

2. Integer Overflow

```
short value = 32767 + 1; // is -32768
```

3. Round-off Errors

```
float a = 1000.43;
float b = 1000.0;
cout << a - b << endl;
```

displays `0.429993`, not `0.43`

Common Errors

4. Unintended Integer Division

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
cout << average << endl;
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
cout << average << endl;
```

(b)

(a) displays 1, (b) displays 1.5

5. Forgetting Header Files

```
#include <cmath>   // needed for pow()
#include <ctime>   // needed for time()
```

Outline

- Writing a Simple Program
- Reading Input from the Keyboard
- Identifiers
- Variables
- Assignment Statements and Assignment Expressions
- Named Constants
- Numeric Data Types and Operations
- Evaluating Expressions and Operator Precedence
- Case Study: Displaying the Current Time
- Augmented Assignment Operators
- Increment and Decrement Operators
- Numeric Type Conversions
- Case Study: Counting Monetary Units
- Common Errors