COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

Mawish.waqas@cuilahore.edu.pk

Chapter 2:
Basic Elements of C++

Objectives

In this chapter, you will:

- Become familiar with the basic components of a C++ program, including functions, special symbols, and identifiers in C++
- Explore simple data types
- Discover how to use arithmetic operators
- Examine how a program evaluates arithmetic expressions
- Become familiar with the `string` data type
- Learn what an assignment statement is and what it does

Objectives (cont’d.)

- Learn about variable declaration
- Discover how to input data into memory using input statements
- Become familiar with the use of increment and decrement operators
- Examine ways to output results using output statements
- Learn how to use preprocessor directives and why they are necessary

Objectives (cont’d.)

- Learn how to debug syntax errors
- Explore how to properly structure a program, including using comments to document a program
- Become familiar with compound statements
- Learn how to write a C++ program

Introduction

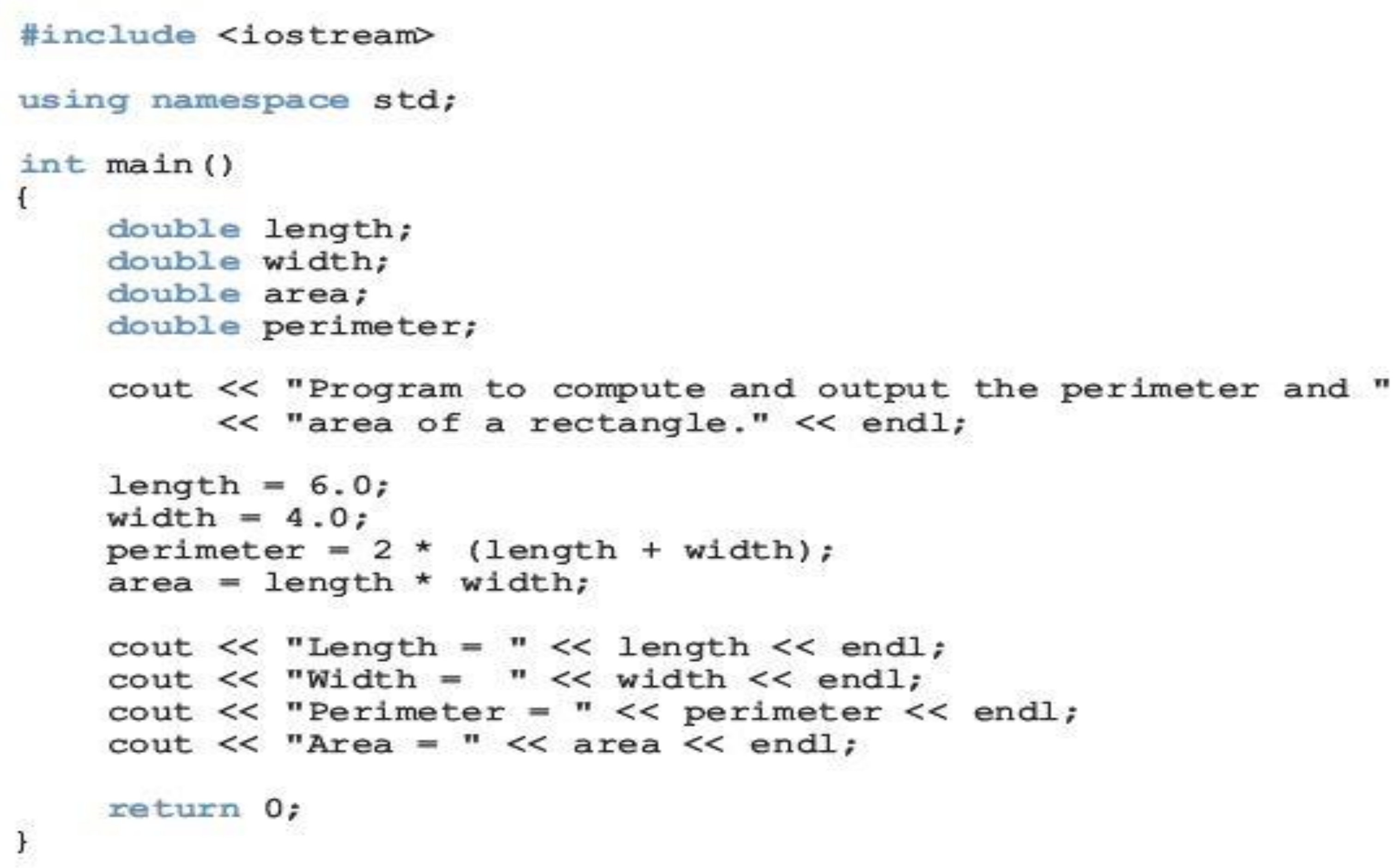
Computer program

- **Sequence of statements** whose objective is to accomplish a task

Programming

- Process of planning and creating a program

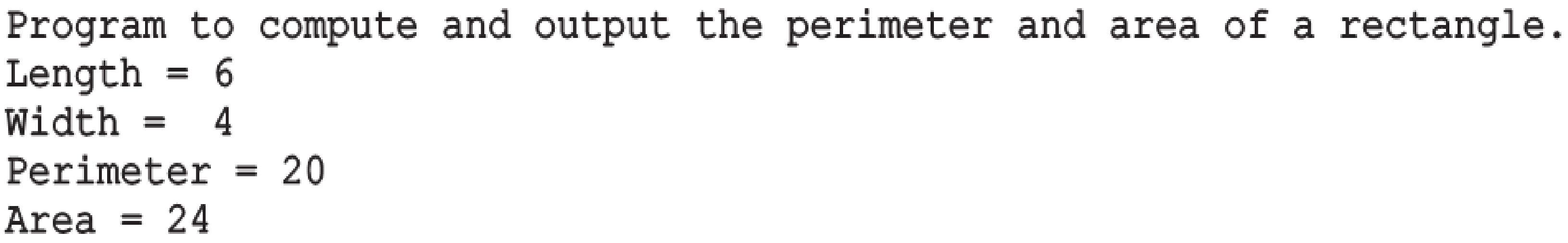
A Quick Look at a C++ Program



```
#include <iostream>

using namespace std;

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

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
    width = 4.0;
    perimeter = 2 * (length + width);
    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

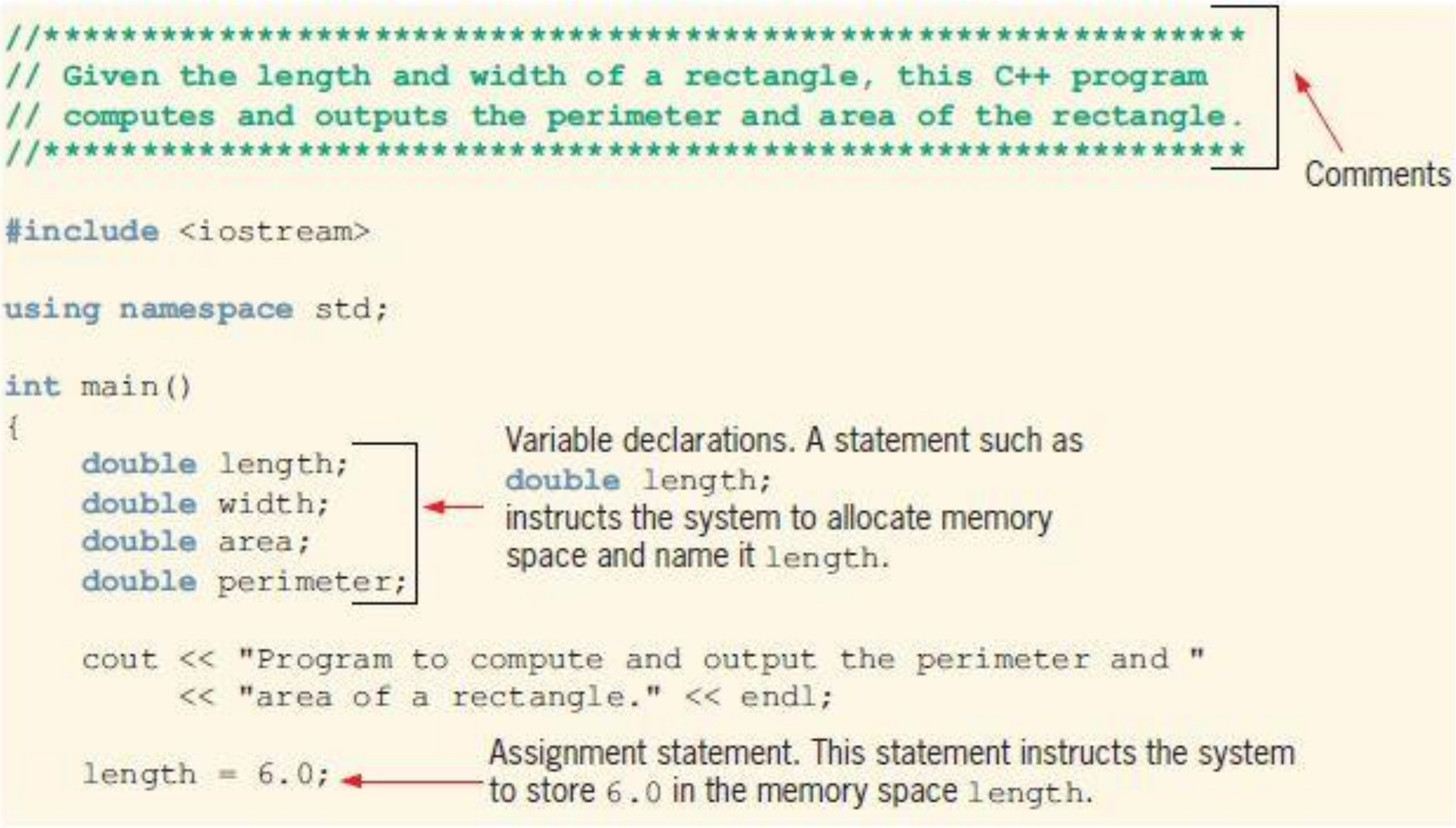
(cont’d.)

Sample run:



```
Program to compute and output the perimeter and area of a rectangle.
Length = 6
Width =  4
Perimeter = 20
Area = 24
```

(cont’d.)



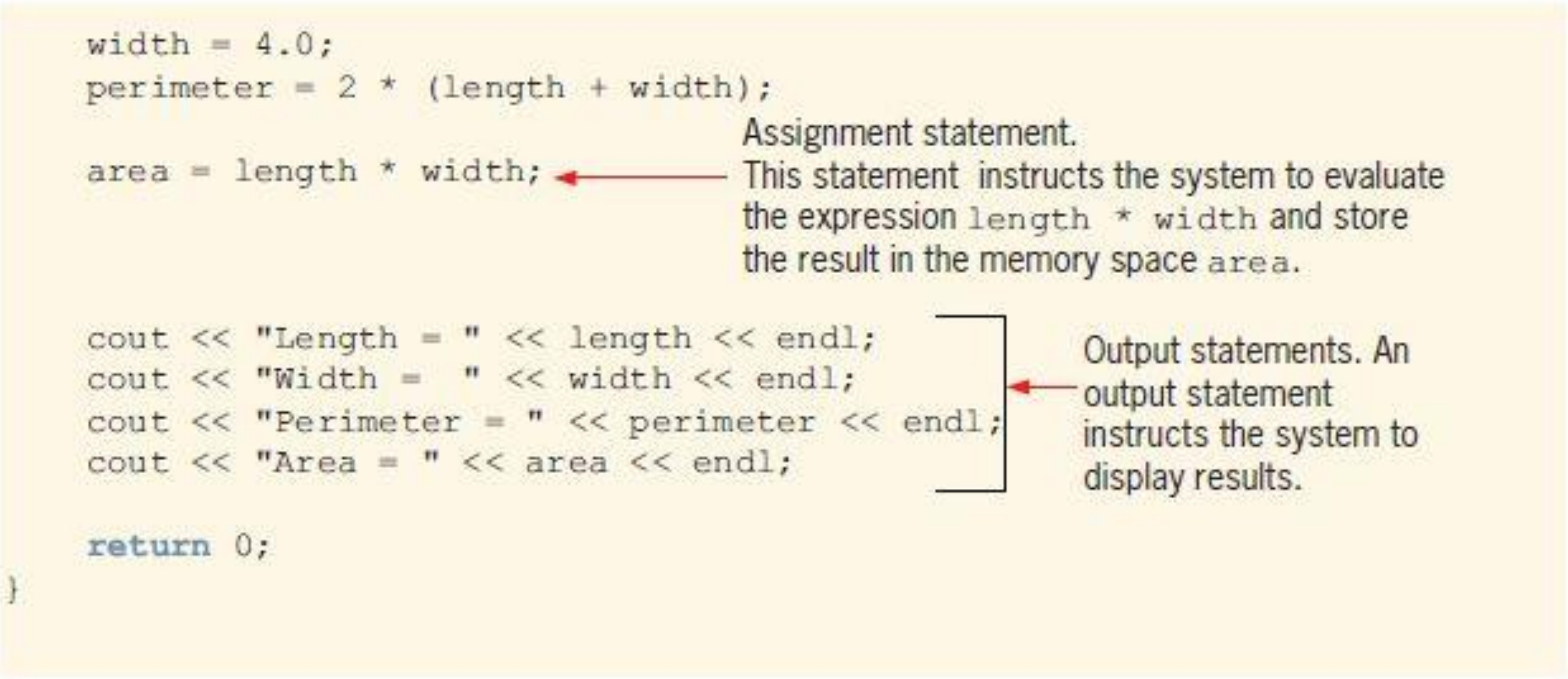
```
//**************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//**************************************************************

#include <iostream>

using namespace std;

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

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
```

Comments

Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it `length`.

Assignment statement. This statement instructs the system to store `6.0` in the memory space `length`.

A Quick Look at a C++ Program(cont’d.)



```
    width = 4.0;
    perimeter = 2 * (length + width);

    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

Assignment statement. This statement instructs the system to evaluate the expression `length * width` and store the result in the memory space `area`.

Output statements. An output statement instructs the system to display results.

FIGURE 2-1 Various parts of a C++ program

A Quick Look at a C++ Program (cont'd.)

Variable: a memory location whose contents can be changed.

- Initially, it contains some random value (called garbage value)

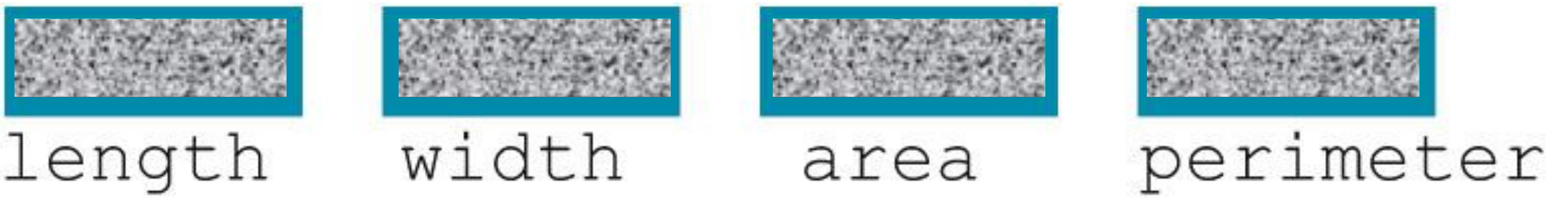


Figure 2-2 Memory allocation

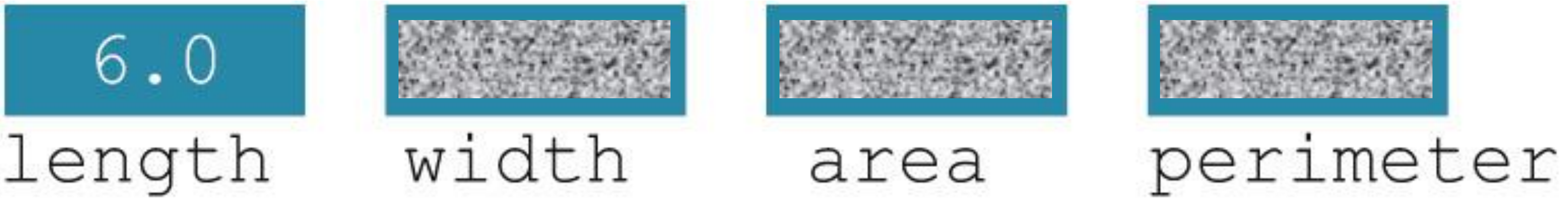


Figure 2-3 Memory spaces after the statement length = 6.0; **executes**

The Basics of a C++ Program

Function (or subprogram): collection of statements; when executed, accomplishes something

- May be predefined or programmer-defined

Syntax rules: rules that specify which statements (instructions) are legal or valid.

Semantic rules: determine the meaning of the instructions.

Programming language: a set of rules, symbols, and special words

Comments

Comments are for the reader, not the compiler

Two types:

- Single line: begin with //

```
// This is a C++ program.
// Welcome to C++ Programming.
```

- Multiple line: enclosed between /* and */

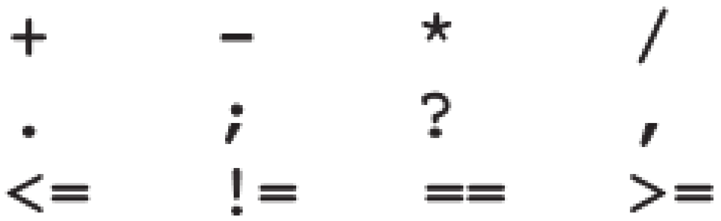
```
/*
   You can include comments that can
   occupy several lines.
*/
```

Special Symbols

Token: the smallest individual unit of a program written in any language, which has a meaning.

C++ tokens include special symbols, reserved word symbols, and identifiers

Special symbols in C++ include (but not limited to):



+	-	*	/
.	;	?	,
<=	!=	==	>=

Reserved Words (Keywords)

Reserved word symbols (or keywords):

- Cannot be redefined within program
- Cannot be used for anything other than their *intended use*

Examples:

- `int`
- `float`
- `double`
- `char`
- `const`
- `void`
- `return`

Identifiers

Identifier: the name of something that appears in a program

- Consists of letters, digits, and the underscore character (_) (**and nothing else**)
- **MUST** begin with a letter or underscore, i.e. cannot begin with a digit

Two predefined identifiers are `cout` and `cin`

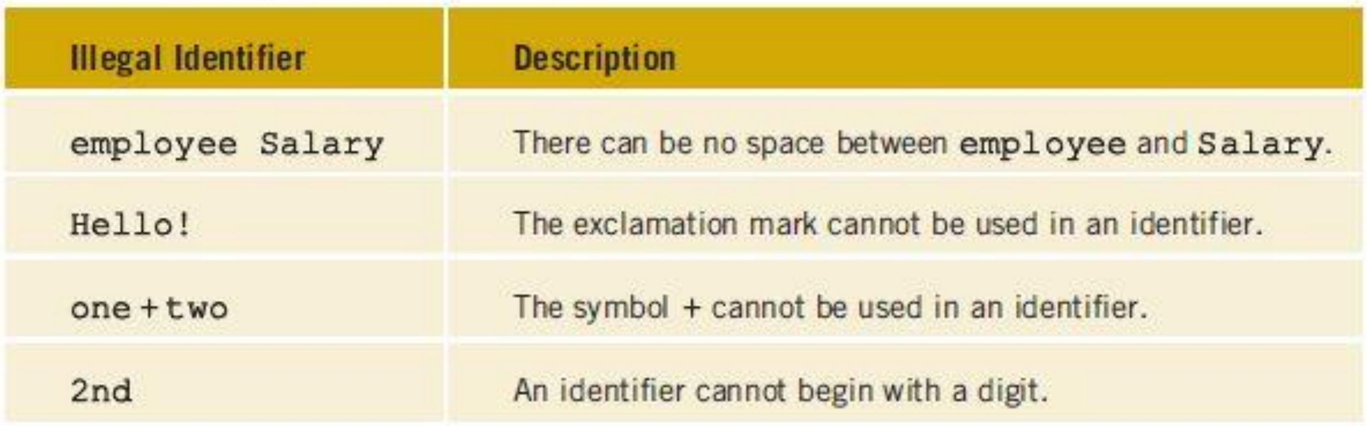
C++ is case sensitive (`COUT` is ***not*** the same as `cout`)

Identifiers (cont'd.)

Legal identifiers in C++:

- `first`
- `c0nversi0n`
- `payRate123`
- `pay_rate`
- `_payRate`

TABLE 2-1 Examples of Illegal Identifiers



Illegal Identifier	Description
`employee Salary`	There can be no space between `employee` and `Salary`.
`Hello!`	The exclamation mark cannot be used in an identifier.
`one+two`	The symbol + cannot be used in an identifier.
`2nd`	An identifier cannot begin with a digit.

Whitespaces

Every C++ program contains whitespaces

- Include blanks, tabs, and newline characters

Used to separate special symbols, reserved words, and identifiers,

- e.g. int marks; // keyword + whitespace + identifier

Proper utilization of whitespaces is important

- E.g. intmarks; // wrong, as no whitespace between keyword and identifier

Whitespace can also be used to make the program more readable (Further details later)

Data Types

Data type:

- Represents Type of data
- set of values together with a set of operations

C++ data types fall into three categories:

- Simple data type
- Structured data type
- Pointers

Simple Data Types

Three categories of simple data

- Integral: integers (numbers without a decimal)
  - Includes the following types:
    - `char, short, int, long, bool, unsigned char, unsigned short, unsigned int, unsigned long`
- Floating-point: decimal numbers
  - Includes the following types:
    - `float, double`
- Enumeration type: user-defined data type

`int` Data Type

Examples of integer constants:

```
-6728
0
78
+763
```

Cannot use a comma within an integer

- Commas are only used for separating items in a list, e.g. int x, y, z;

`bool` Data Type

`bool` type

- Two values: `true` and `false`
- Example: male/female, pass/fail etc.
- Manipulate logical (Boolean) expressions

`true` and `false`

- Logical(Boolean) values

`bool`, `true`, and `false`

- Reserved words

`char` Data Type

- The smallest integral data type
- Used for single characters: letters, digits, and special symbols
- Each character is enclosed in single quotes,
- Examples of character constants:
  - `‘A’, 'a', '0', '*', '+', '$', '&'`
- A blank space is a character
  - Written `'  '`, with a space left between the single quotes
- Note that ‘3’ and 3 are char and int, respectively, and are NOT the same.

`char` Data Type (cont'd.)

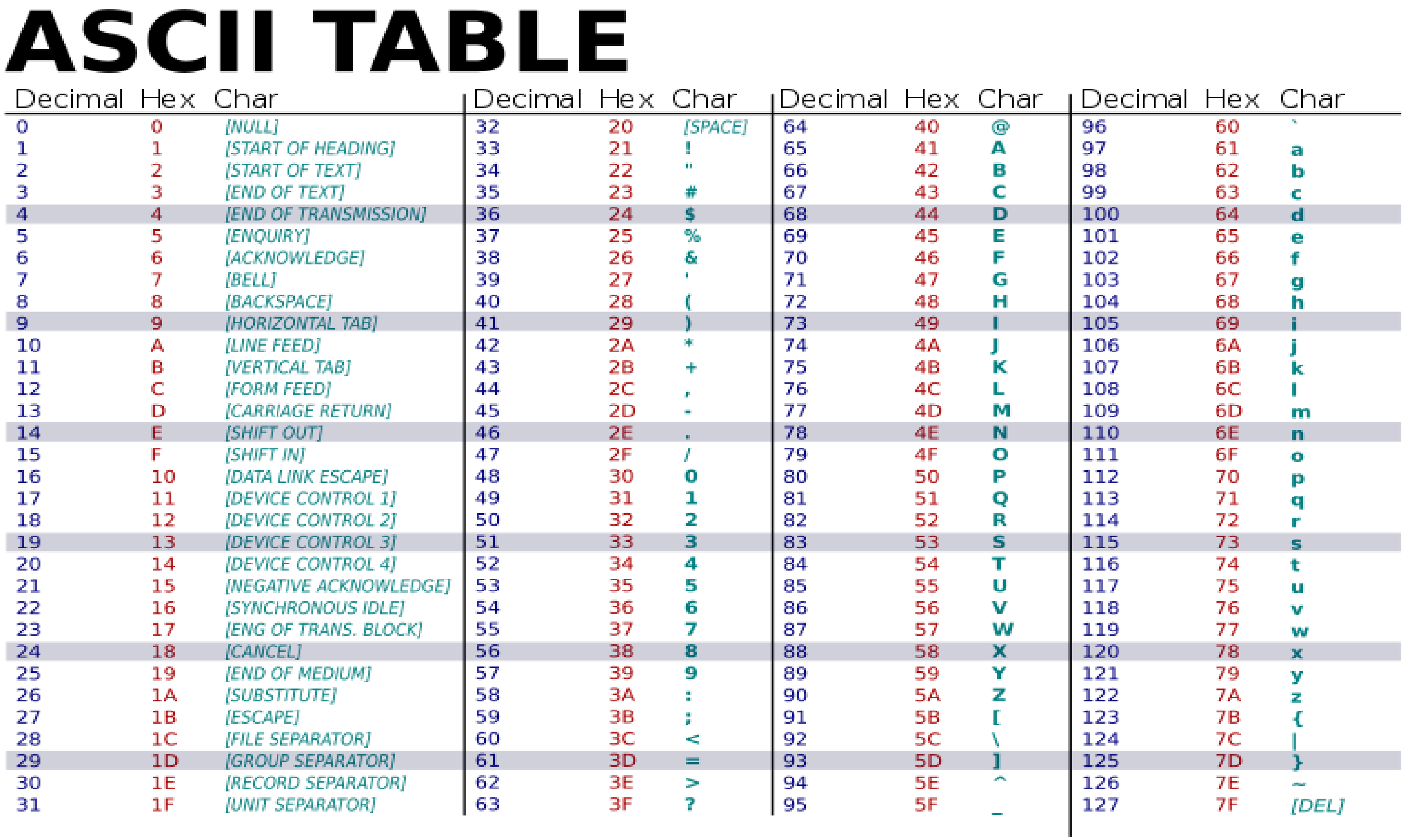
Different character data sets exist

ASCII: American Standard Code for Information Interchange

- Each of 128 values in ASCII code set represents a different character
- Characters have a predefined ordering based on the ASCII numeric value

Collating sequence: ordering of characters based on the character set code, so 2 characters can be compared based on this order, e.g. 'A' is greater than '0'

Complete ASCII Character Set



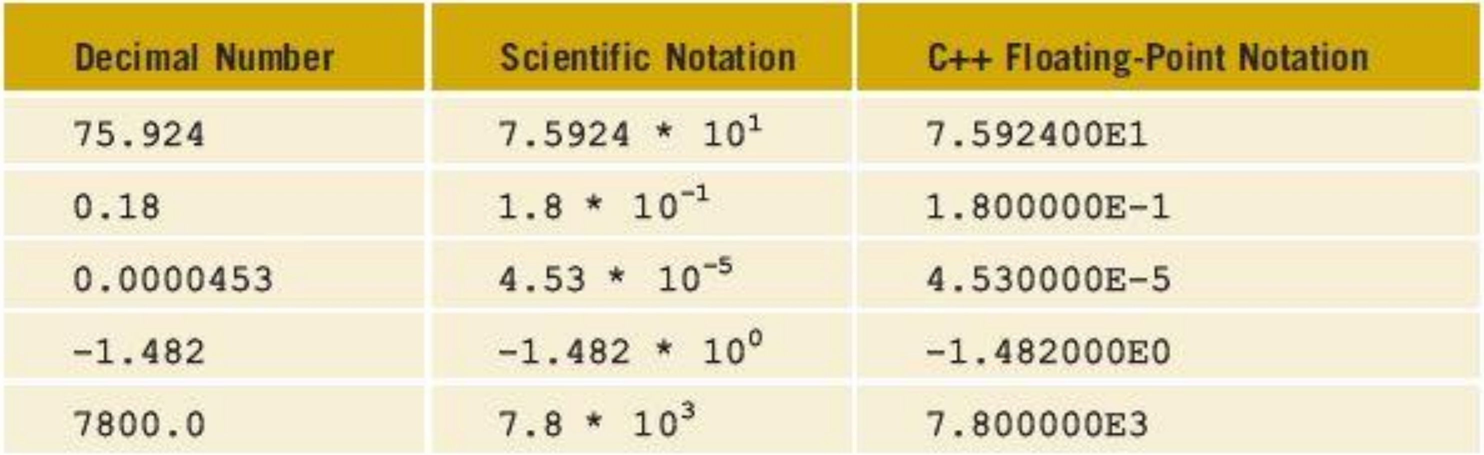
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Floating-Point Data Types

C++ uses scientific notation to represent real numbers (floating-point notation)

TABLE 2-3 Examples of Decimal Numbers in Scientific and C++ Floating-Point Notations



Decimal Number	Scientific Notation	C++ Floating-Point Notation
75.924	7.5924 * 10^{1}	7.592400E1
0.18	1.8 * 10^{-1}	1.800000E-1
0.0000453	4.53 * 10^{-5}	4.530000E-5
-1.482	-1.482 * 10^{0}	-1.482000E0
7800.0	7.8 * 10^{3}	7.800000E3

Floating-Point Data Types (cont'd.)

`float`: represents any real number

- Range: -3.4E+38 to 3.4E+38 (four bytes)

`double`: represents any real number

- Range: -1.7E+308 to 1.7E+308 (eight bytes)

Minimum and maximum values of data types are system (compiler + OS) dependent

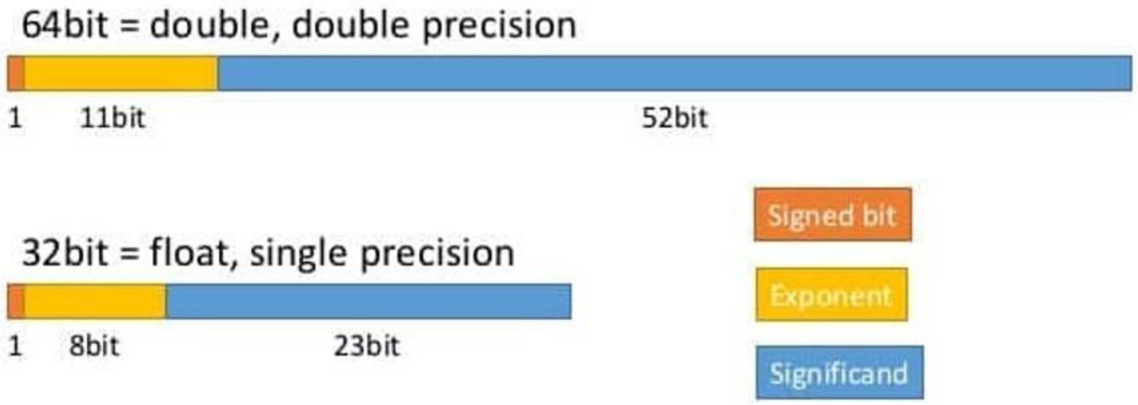
Floating-Point Data Types (cont'd.)

Maximum number of significant digits (decimal places) for `float` values: 6 or 7

Maximum number of significant digits for `double`: 15

<u>Precision</u>: maximum number of significant digits

- Float values are called <u>single precision</u>
- Double values are called <u>double precision</u>


64bit = double, double precision
1
11bit
52bit
32bit = float, single precision
1
8bit
23bit
Signed bit
Exponent
Significand

`string` Type

- Programmer-defined type supplied in ANSI/ISO Standard C++ library
- Sequence of zero or more characters enclosed in double quotation marks
  - Example: “Hello World”, “Pakistan”, “A”
- Null (or empty): a string with no characters,
  - Example: “” // empty string
- Length of a string is number of characters in it (more about this later)
  - Example: length of "`William Jacob`" is 13

Simple Data Types (cont’d.)

- The following values may vary on your compiler.

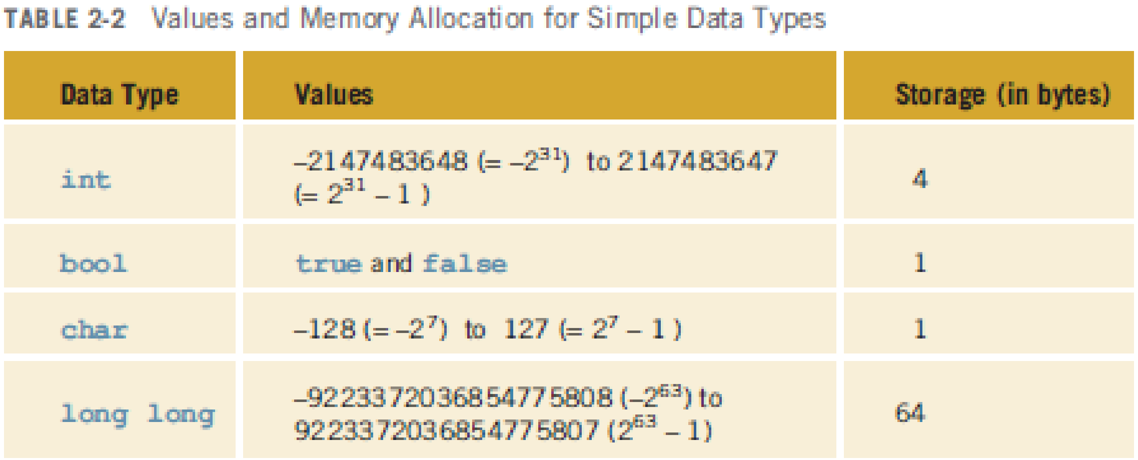
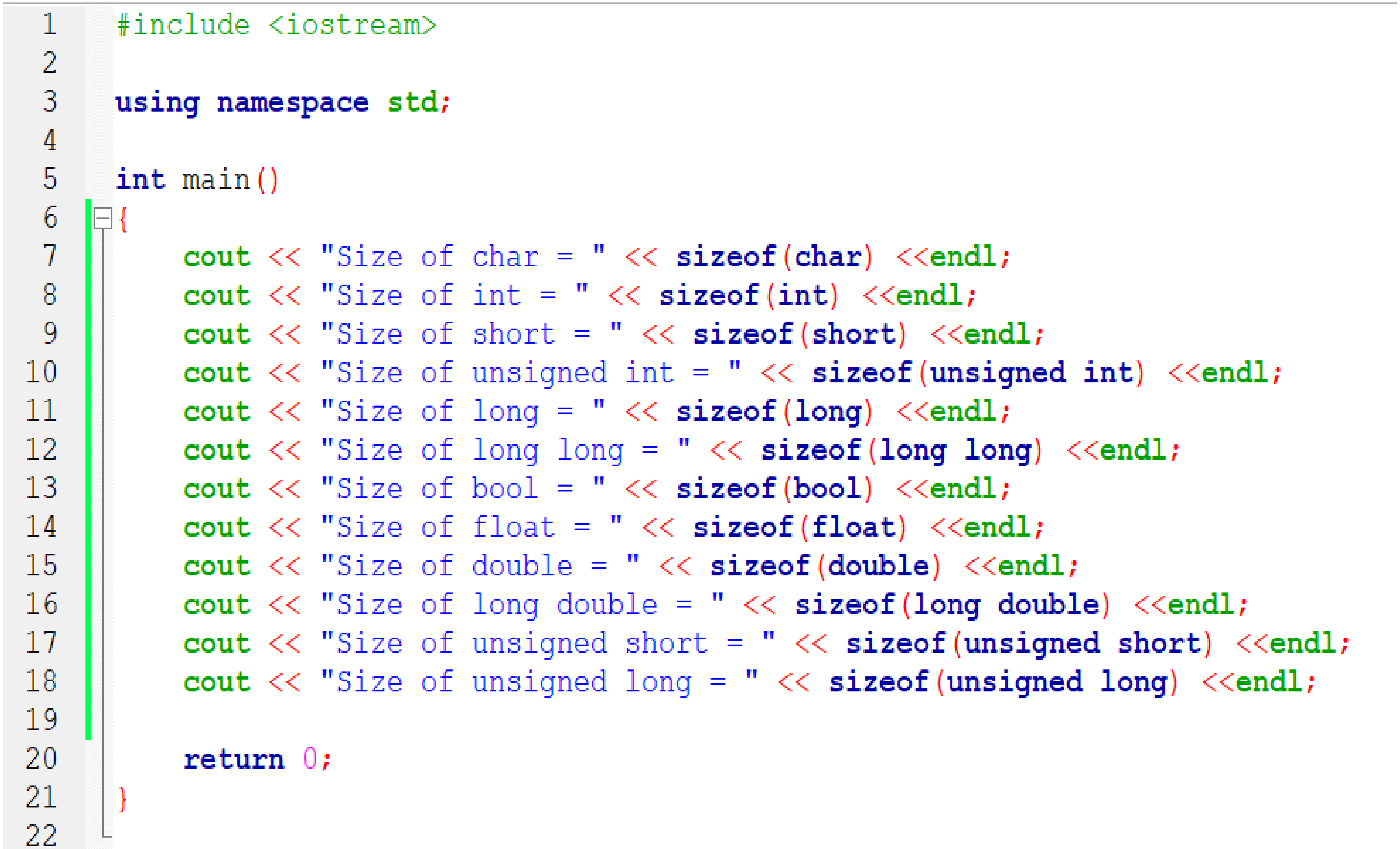


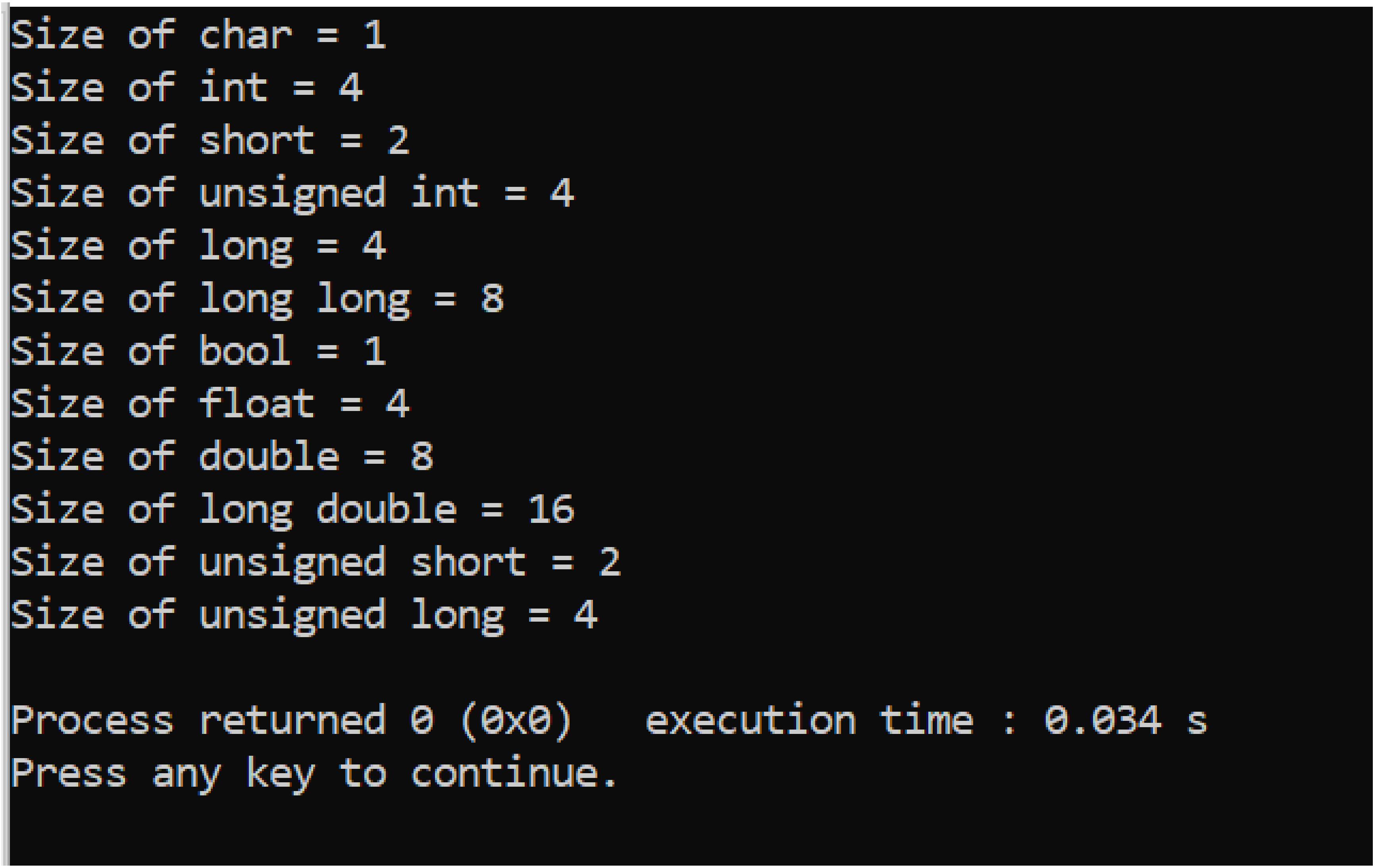
TABLE 2-2 Values and Memory Allocation for Simple Data Types

Data Type	Values	Storage (in bytes)
`int`	-2147483648 (= -2^{31}) to 2147483647 (= $2^{31} - 1$)	4
`bool`	`true` and `false`	1
`char`	-128 (= -2^{7}) to 127 (= $2^{7} - 1$)	1
`long long`	-9223372036854775808 (-2^{63}) to 9223372036854775807 ($2^{63} - 1$)	64

- Different compilers may allow different ranges of values depending on the size of type (in bytes)
  - Check the size of some type using the sizeof operator.
  - Example: cout << sizeof(int) << “ “ << sizeof(double);
  - Example: cout << sizeof(float) << “ “ << sizeof(long double);



```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Size of char = " << sizeof(char) <<endl;
    cout << "Size of int = " << sizeof(int) <<endl;
    cout << "Size of short = " << sizeof(short) <<endl;
    cout << "Size of unsigned int = " << sizeof(unsigned int) <<endl;
    cout << "Size of long = " << sizeof(long) <<endl;
    cout << "Size of long long = " << sizeof(long long) <<endl;
    cout << "Size of bool = " << sizeof(bool) <<endl;
    cout << "Size of float = " << sizeof(float) <<endl;
    cout << "Size of double = " << sizeof(double) <<endl;
    cout << "Size of long double = " << sizeof(long double) <<endl;
    cout << "Size of unsigned short = " << sizeof(unsigned short) <<endl;
    cout << "Size of unsigned long = " << sizeof(unsigned long) <<endl;

    return 0;
}

```



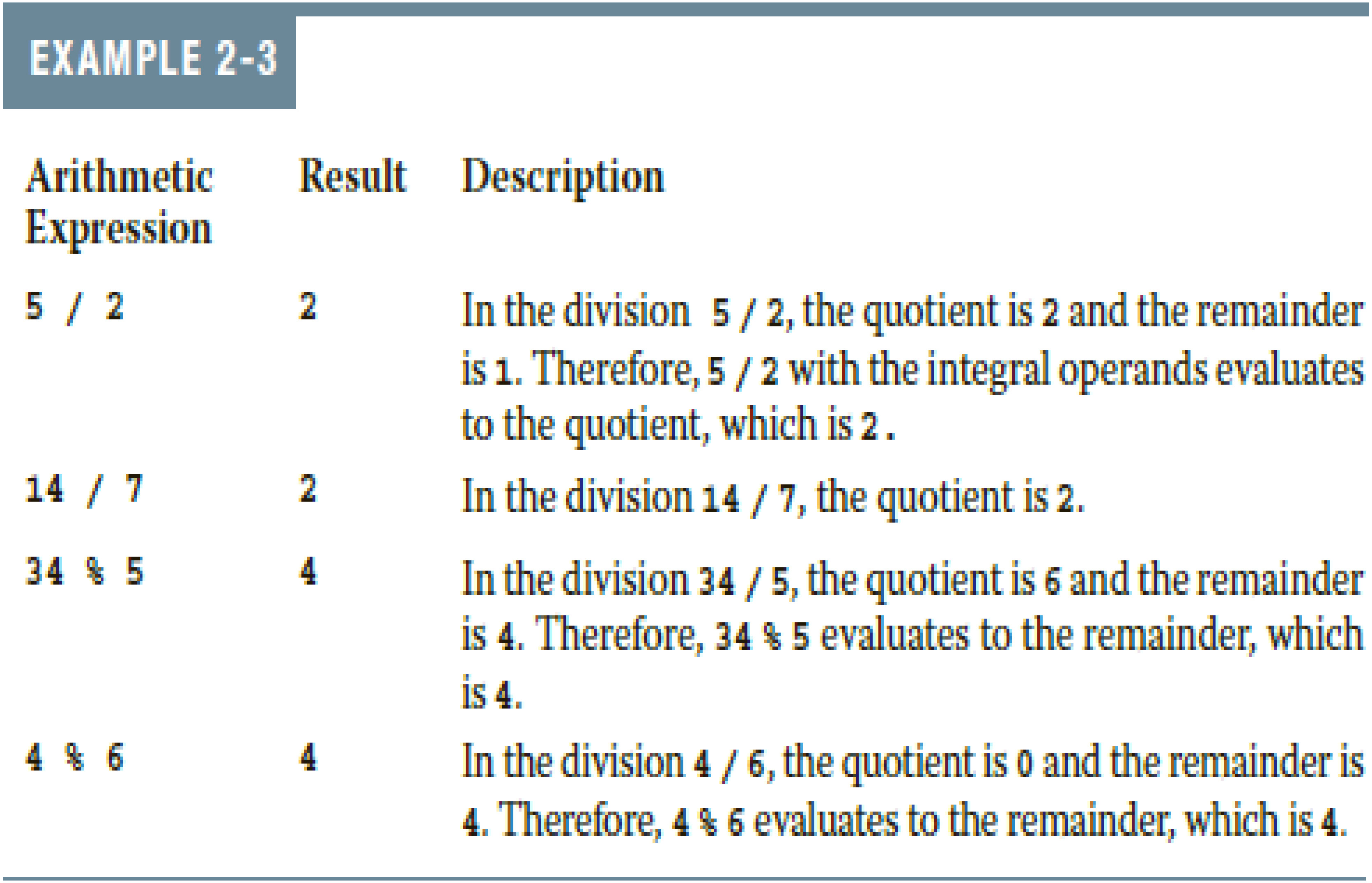
```
Size of char = 1
Size of int = 4
Size of short = 2
Size of unsigned int = 4
Size of long = 4
Size of long long = 8
Size of bool = 1
Size of float = 4
Size of double = 8
Size of long double = 16
Size of unsigned short = 2
Size of unsigned long = 4

Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```

Arithmetic Operators, Operator Precedence, and Expressions

- C++ arithmetic operators:
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % modulus (or remainder) operator
- +, -, *, and / can be used with integral and floating-point data types
- Use % only with integral data types

Arithmetic Operators, Operator Precedence, and Expressions (cont'd.)

- When you use / with integral data types, the integral result is truncated (no rounding)
  - 5/2 = 2
- Arithmetic expressions: contain values and arithmetic operators
- Operands: the number of values on which the operators will work
- Operators can be unary (one operand) or binary (two operands)
  - -5



EXAMPLE 2-3

Arithmetic Expression	Result	Description
`5 / 2`	`2`	In the division `5 / 2`, the quotient is `2` and the remainder is `1`. Therefore, `5 / 2` with the integral operands evaluates to the quotient, which is `2`.
`14 / 7`	`2`	In the division `14 / 7`, the quotient is `2`.
`34 % 5`	`4`	In the division `34 / 5`, the quotient is `6` and the remainder is `4`. Therefore, `34 % 5` evaluates to the remainder, which is `4`.
`4 % 6`	`4`	In the division `4 / 6`, the quotient is `0` and the remainder is `4`. Therefore, `4 % 6` evaluates to the remainder, which is `4`.

Order of Precedence

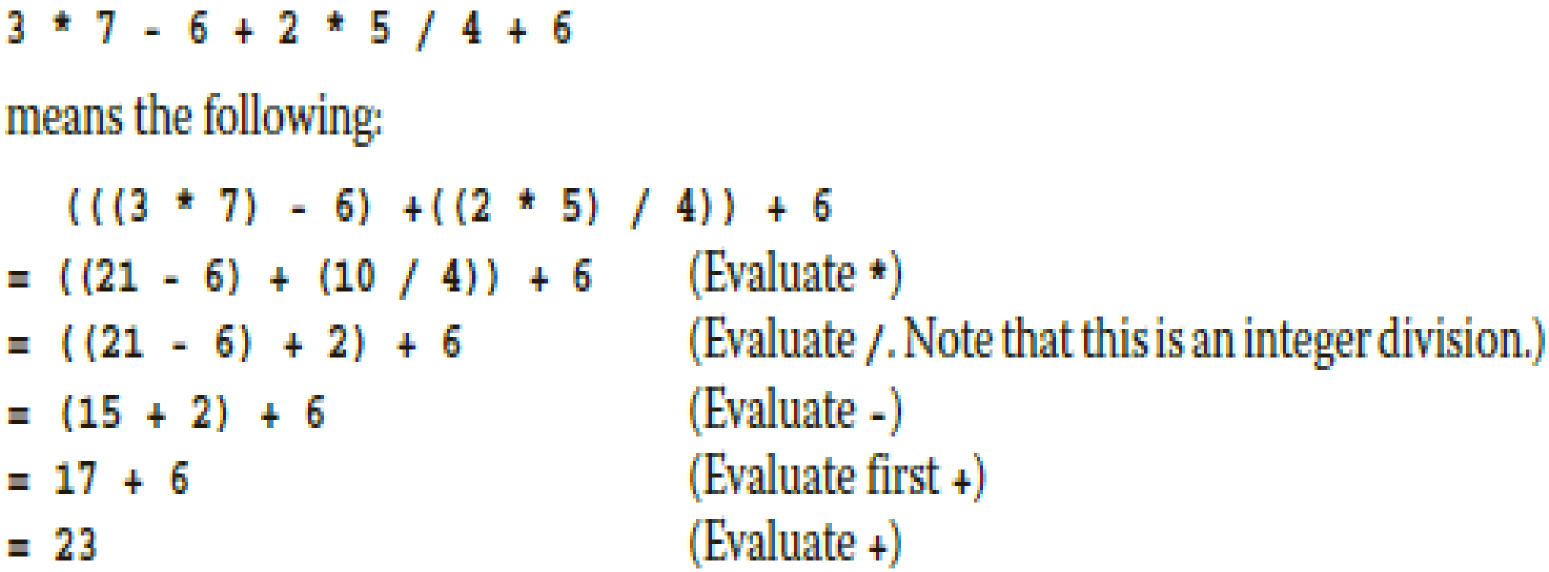
- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- Associativity: all arithmetic operators have left to right associativity

- When operators are on the same level
  - Performed from left to right (associativity)

- `3 * 7 - 6 + 2 * 5 / 4 + 6` means

```
  (((3 * 7) – 6) + ((2 * 5) / 4 )) + 6
```

Order of Precedence



```
3 * 7 - 6 + 2 * 5 / 4 + 6
```

means the following:

```
   (((3 * 7) - 6) +((2 * 5) / 4)) + 6
= ((21 - 6) + (10 / 4)) + 6      (Evaluate *)
= ((21 - 6) + 2) + 6             (Evaluate /. Note that this is an integer division.)
= (15 + 2) + 6                   (Evaluate -)
= 17 + 6                         (Evaluate first +)
= 23                             (Evaluate +)
```

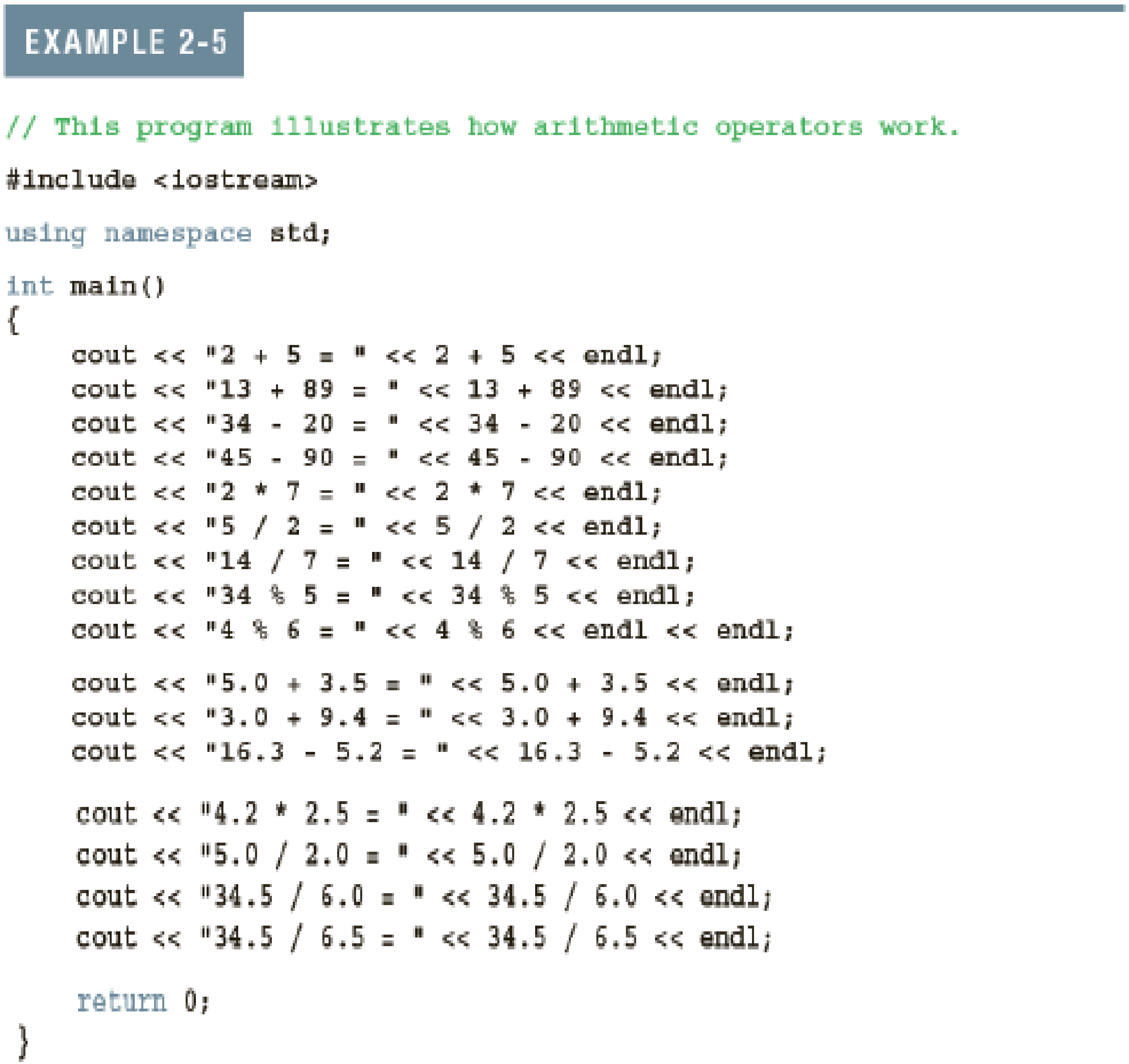
Expressions

Integral expression: all operands are integers

- Yields an integral result
- Example: 5/2 + 3 * 5 = 17

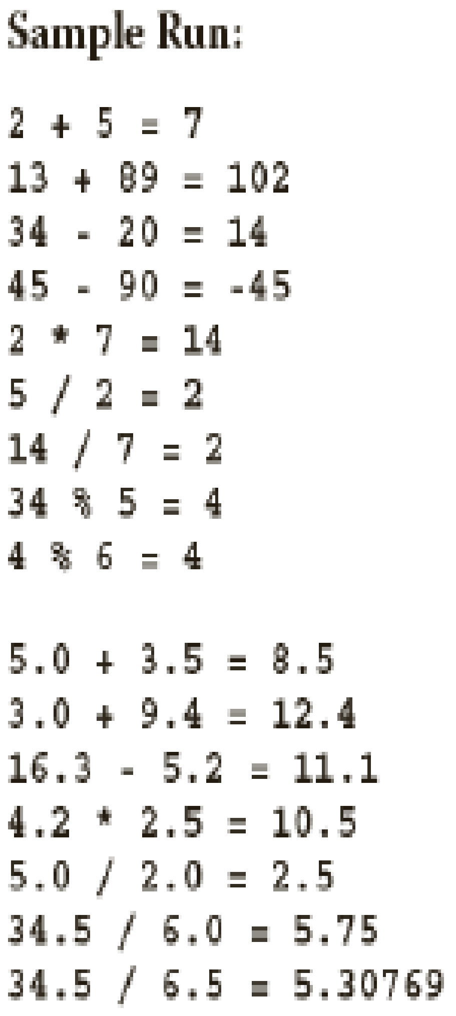
Floating-point expression: all operands are floating-point

- Yields a floating-point result
- Example: 5.0/2.0 + 12.8 * 17.5 - 34.50
- Example: 5/2.0 → 5.0/2.0 → 2.5



EXAMPLE 2-5

```
// This program illustrates how arithmetic operators work.

#include <iostream>

using namespace std;

int main()
{
    cout << "2 + 5 = " << 2 + 5 << endl;
    cout << "13 + 89 = " << 13 + 89 << endl;
    cout << "34 - 20 = " << 34 - 20 << endl;
    cout << "45 - 90 = " << 45 - 90 << endl;
    cout << "2 * 7 = " << 2 * 7 << endl;
    cout << "5 / 2 = " << 5 / 2 << endl;
    cout << "14 / 7 = " << 14 / 7 << endl;
    cout << "34 % 5 = " << 34 % 5 << endl;
    cout << "4 % 6 = " << 4 % 6 << endl << endl;

    cout << "5.0 + 3.5 = " << 5.0 + 3.5 << endl;
    cout << "3.0 + 9.4 = " << 3.0 + 9.4 << endl;
    cout << "16.3 - 5.2 = " << 16.3 - 5.2 << endl;

    cout << "4.2 * 2.5 = " << 4.2 * 2.5 << endl;
    cout << "5.0 / 2.0 = " << 5.0 / 2.0 << endl;
    cout << "34.5 / 6.0 = " << 34.5 / 6.0 << endl;
    cout << "34.5 / 6.5 = " << 34.5 / 6.5 << endl;

    return 0;
}
```



Sample Run:

```
2 + 5 = 7
13 + 89 = 102
34 - 20 = 14
45 - 90 = -45
2 * 7 = 14
5 / 2 = 2
14 / 7 = 2
34 % 5 = 4
4 % 6 = 4

5.0 + 3.5 = 8.5
3.0 + 9.4 = 12.4
16.3 - 5.2 = 11.1
4.2 * 2.5 = 10.5
5.0 / 2.0 = 2.5
34.5 / 6.0 = 5.75
34.5 / 6.5 = 5.30769
```

Mixed Expressions

Mixed expression:

- Has operands of different data types
- Contains integers and floating-point

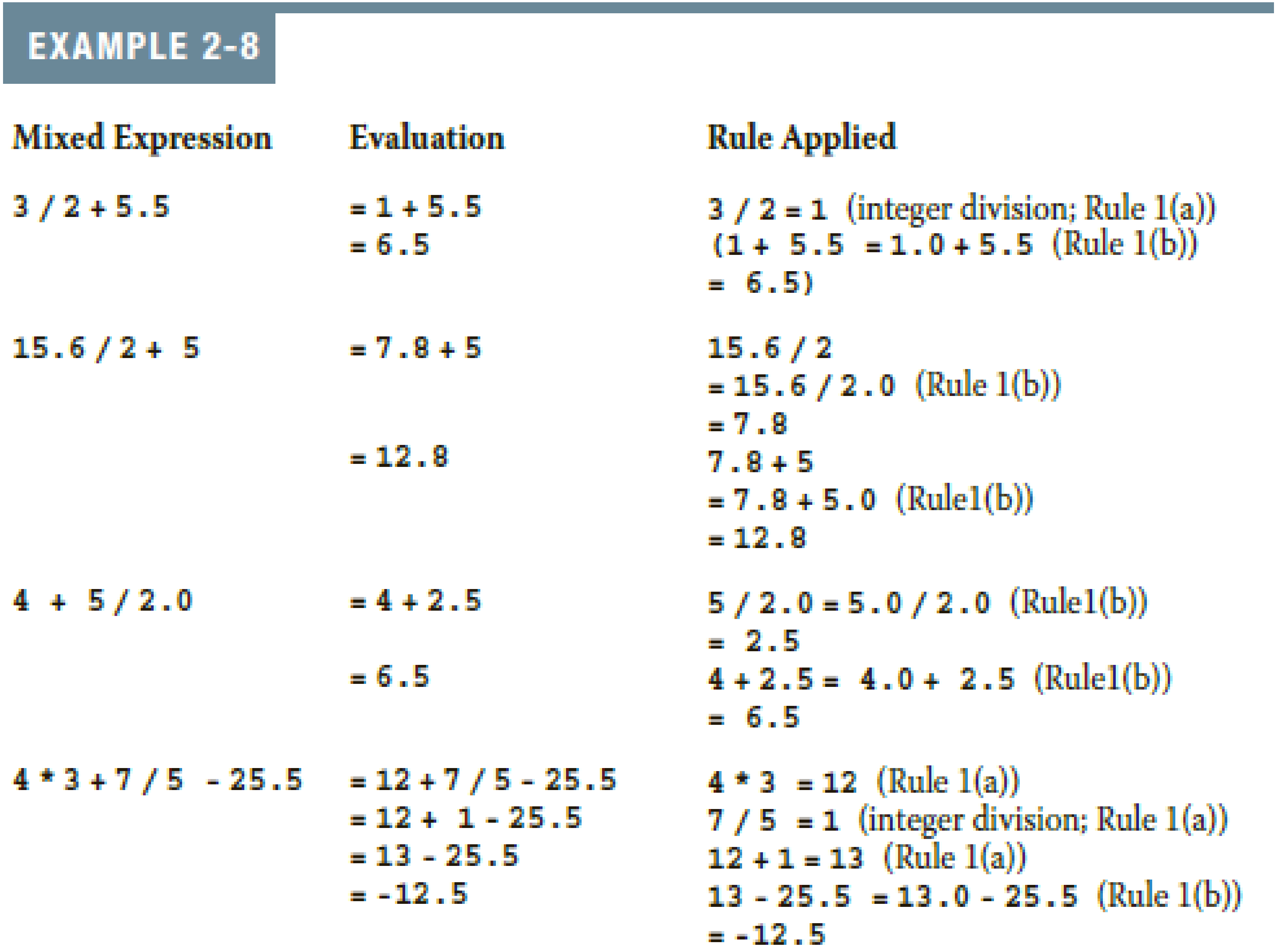
Examples of mixed expressions:

```
2 + 3.5
6  /  4 + 3.9
5.4  *  2 - 13.6 + 18  /  2
```

Mixed Expressions (cont’d.)

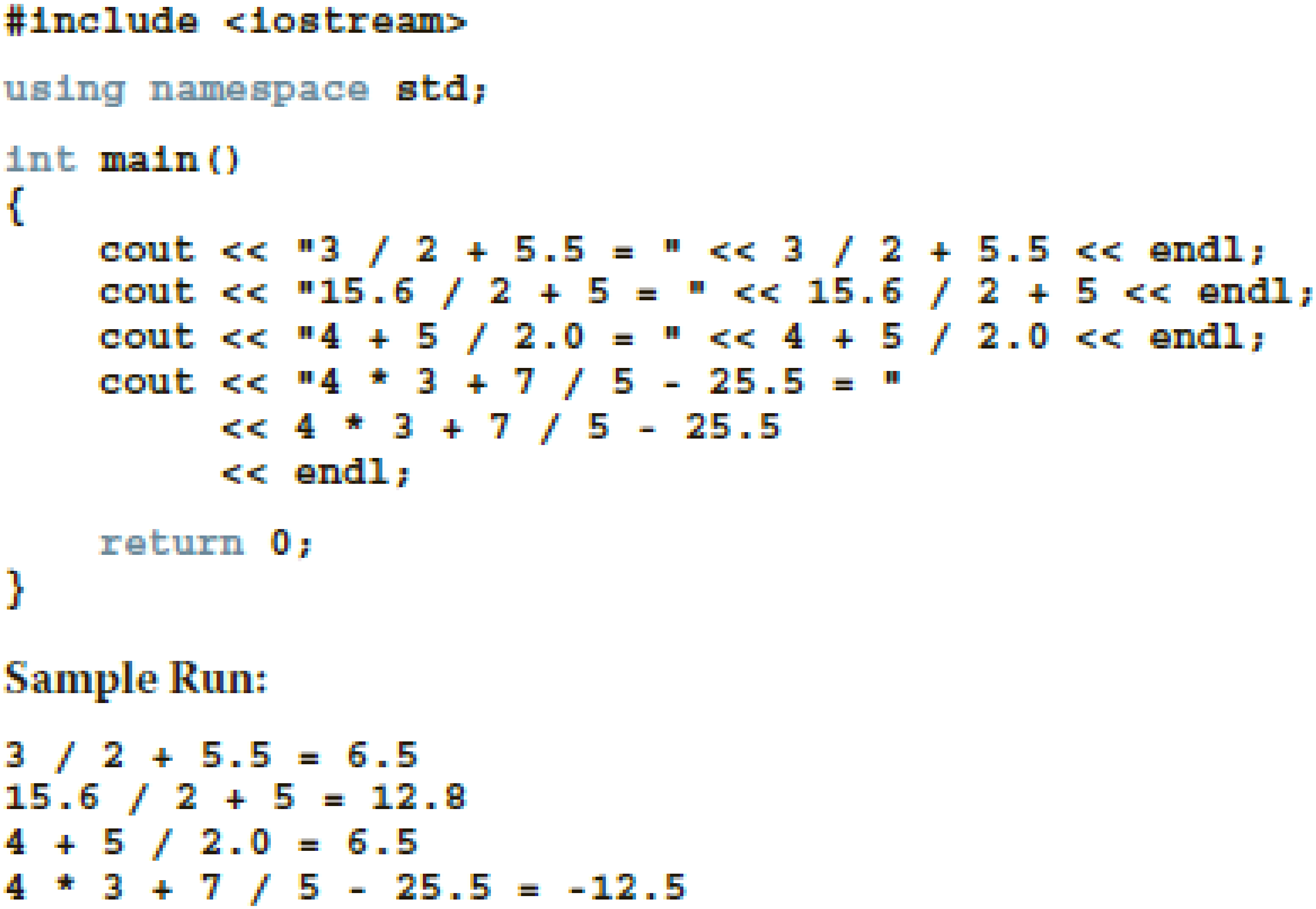
Evaluation rules:

- If operator has same types of operands
  - Evaluated according to the type of the operands
- If operator has both types of operands, e.g. 5/2.0
  - Integer is changed to floating-point
  - Operator is evaluated
  - Result is floating-point
- Entire expression is evaluated according to precedence rules



EXAMPLE 2-8

Mixed Expression	Evaluation	Rule Applied
3 / 2 + 5.5	= 1 + 5.5 = 6.5	3 / 2 = 1 (integer division; Rule 1(a)) (1 + 5.5 = 1.0 + 5.5 (Rule 1(b)) = 6.5)
15.6 / 2 + 5	= 7.8 + 5 = 12.8	15.6 / 2 = 15.6 / 2.0 (Rule 1(b)) = 7.8 7.8 + 5 = 7.8 + 5.0 (Rule1(b)) = 12.8
4 + 5 / 2.0	= 4 + 2.5 = 6.5	5 / 2.0 = 5.0 / 2.0 (Rule1(b)) = 2.5 4 + 2.5 = 4.0 + 2.5 (Rule1(b)) = 6.5
4 * 3 + 7 / 5 - 25.5	= 12 + 7 / 5 - 25.5 = 12 + 1 - 25.5 = 13 - 25.5 = -12.5	4 * 3 = 12 (Rule 1(a)) 7 / 5 = 1 (integer division; Rule 1(a)) 12 + 1 = 13 (Rule 1(a)) 13 - 25.5 = 13.0 - 25.5 (Rule 1(b)) = -12.5



```
#include <iostream>

using namespace std;

int main()
{
    cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
    cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
    cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
    cout << "4 * 3 + 7 / 5 - 25.5 = "
         << 4 * 3 + 7 / 5 - 25.5
         << endl;

    return 0;
}
```

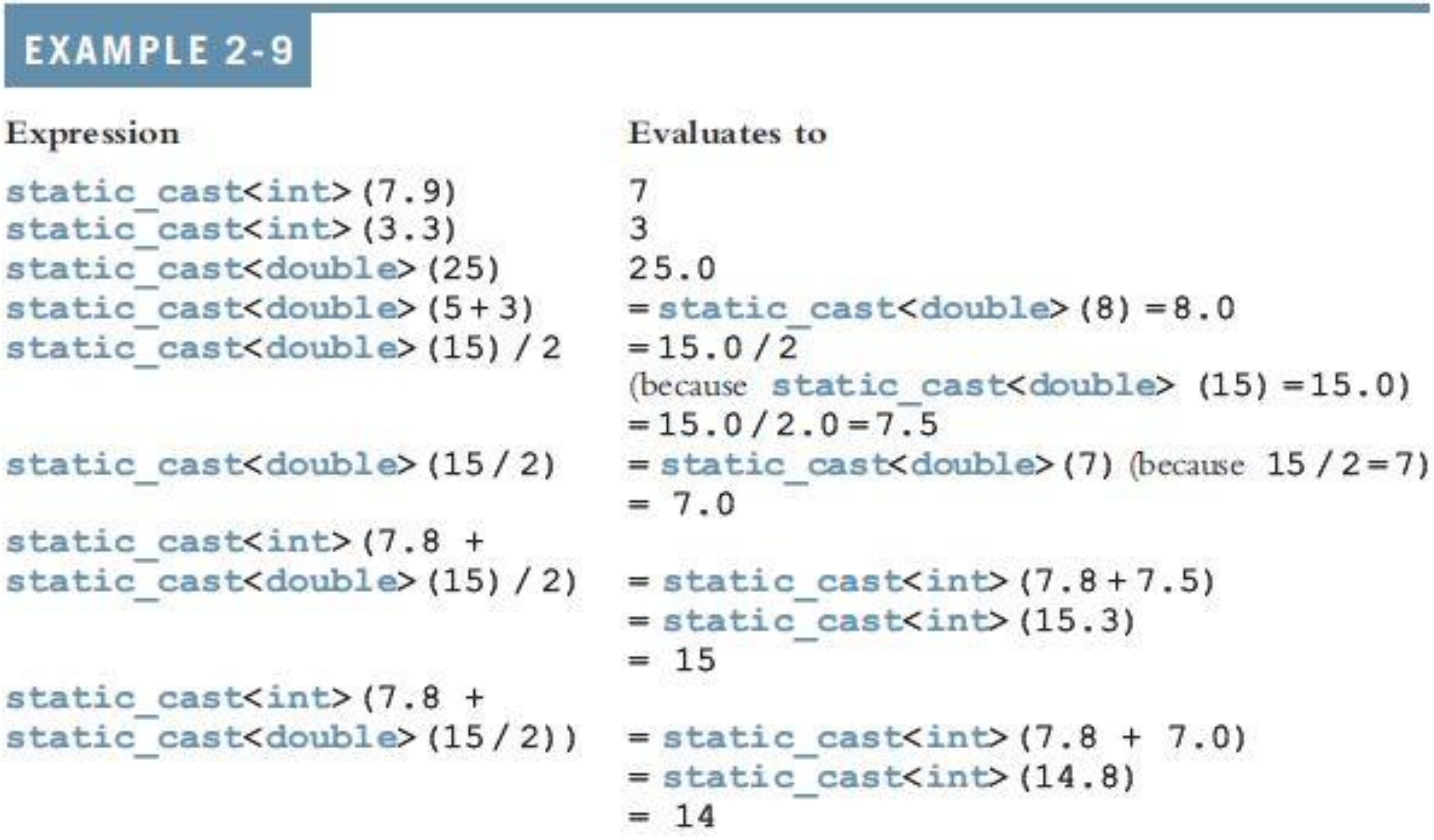
Sample Run:

```
3 / 2 + 5.5 = 6.5
15.6 / 2 + 5 = 12.8
4 + 5 / 2.0 = 6.5
4 * 3 + 7 / 5 - 25.5 = -12.5
```

Type Conversion (Casting)

- Implicit type conversion: when value of one type is automatically changed to another type
- Cast operator: provides explicit type conversion
  - `static_cast<dataTypeName>(expression)`
  - `static_cast<double>(x)/y`

Type Conversion (Casting) (cont’d.)



EXAMPLE 2-9

Expression	Evaluates to
static_cast<int>(7.9)	7
static_cast<int>(3.3)	3
static_cast<double>(25)	25.0
static_cast<double>(5+3)	=static_cast<double>(8) =8.0
static_cast<double>(15) / 2	=15.0 / 2 (because static_cast<double> (15) =15.0) =15.0/2.0=7.5
static_cast<double>(15 / 2)	=static_cast<double>(7) (because 15 / 2=7) = 7.0
static_cast<int>(7.8 + static_cast<double>(15) / 2)	= static_cast<int>(7.8+7.5) = static_cast<int>(15.3) = 15
static_cast<int>(7.8 + static_cast<double>(15/2))	= static_cast<int>(7.8 + 7.0) = static_cast<int>(14.8) = 14

Data Types, Variables, and Assignment Statements

- To declare a variable, must specify its data type
- Syntax: `dataType identifier;`
- Examples (declaring variables):

```
int counter; // an integer variable
double interestRate; // a double var
char grade; // a character variable
string cityName; // a string variable
```

- Assignment statement: variable = expression

```
interestRate = 0.05;
cityName = “Lahore”;
```

Variables, Assignment Statements, and Input Statements

Data must be loaded into main memory before it can be manipulated

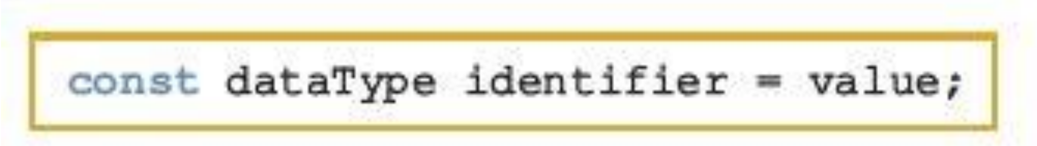
Storing data in memory is a two-step process:

- Instruct computer to allocate memory
- Include statements to put data into memory

Allocating Memory with Constants and Variables

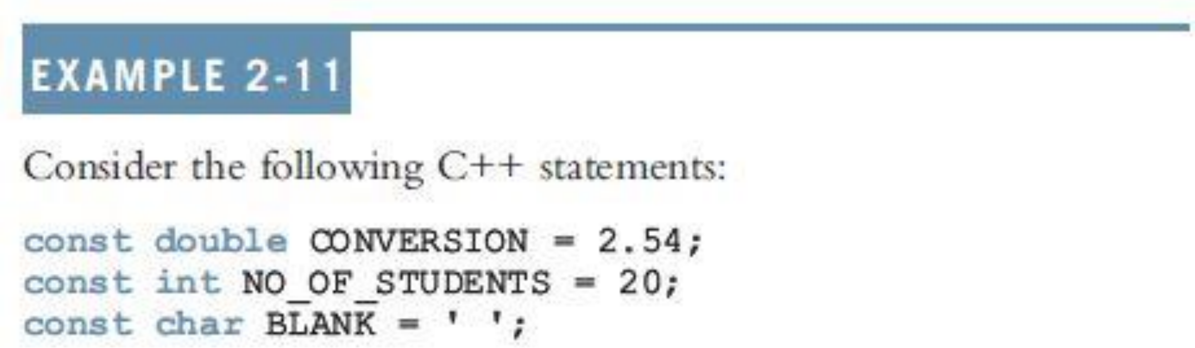
Named constant: memory location whose content can’t change during execution

Syntax to declare a named constant:



```
const dataType identifier = value;
```

In C++, `const` is a reserved word



EXAMPLE 2-11

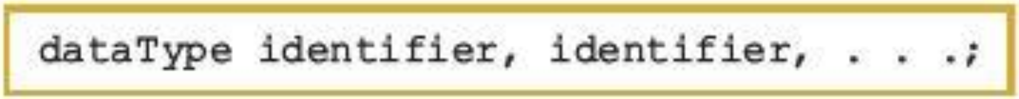
Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

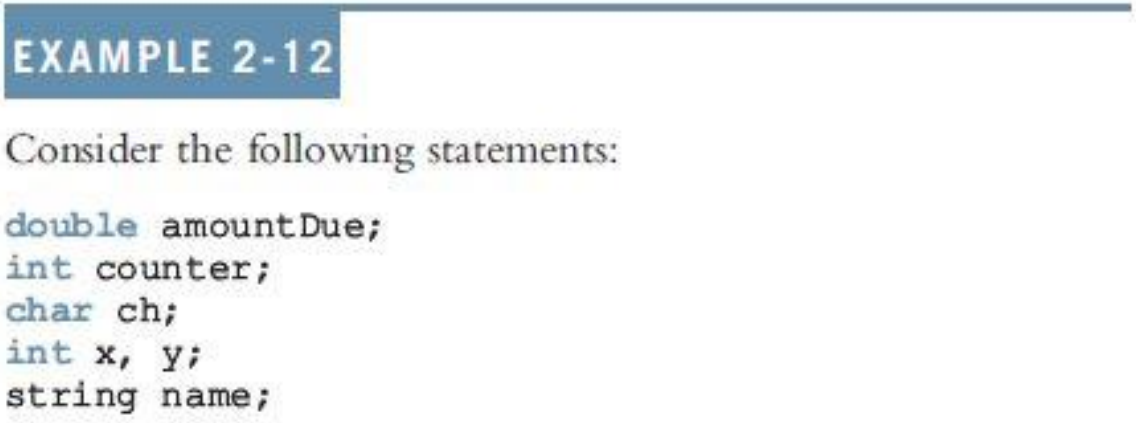
Allocating Memory with Constants and Variables (cont'd.)

Variable: memory location whose content may change during execution

Syntax to declare a named constant:



```
dataType identifier, identifier, . . .;
```



EXAMPLE 2-12

Consider the following statements:

```
double amountDue;
int counter;
char ch;
int x, y;
string name;
```

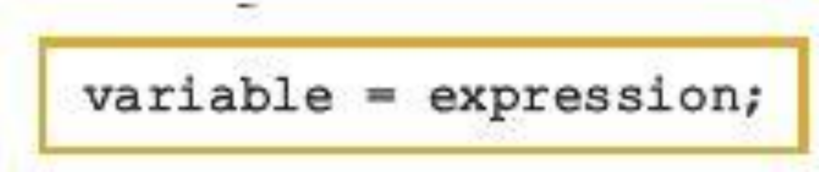
Putting Data into Variables

Ways to place data into a variable:

- Use C++’s assignment statement
- Use input (read) statements

Assignment Statement

The assignment statement takes the form:



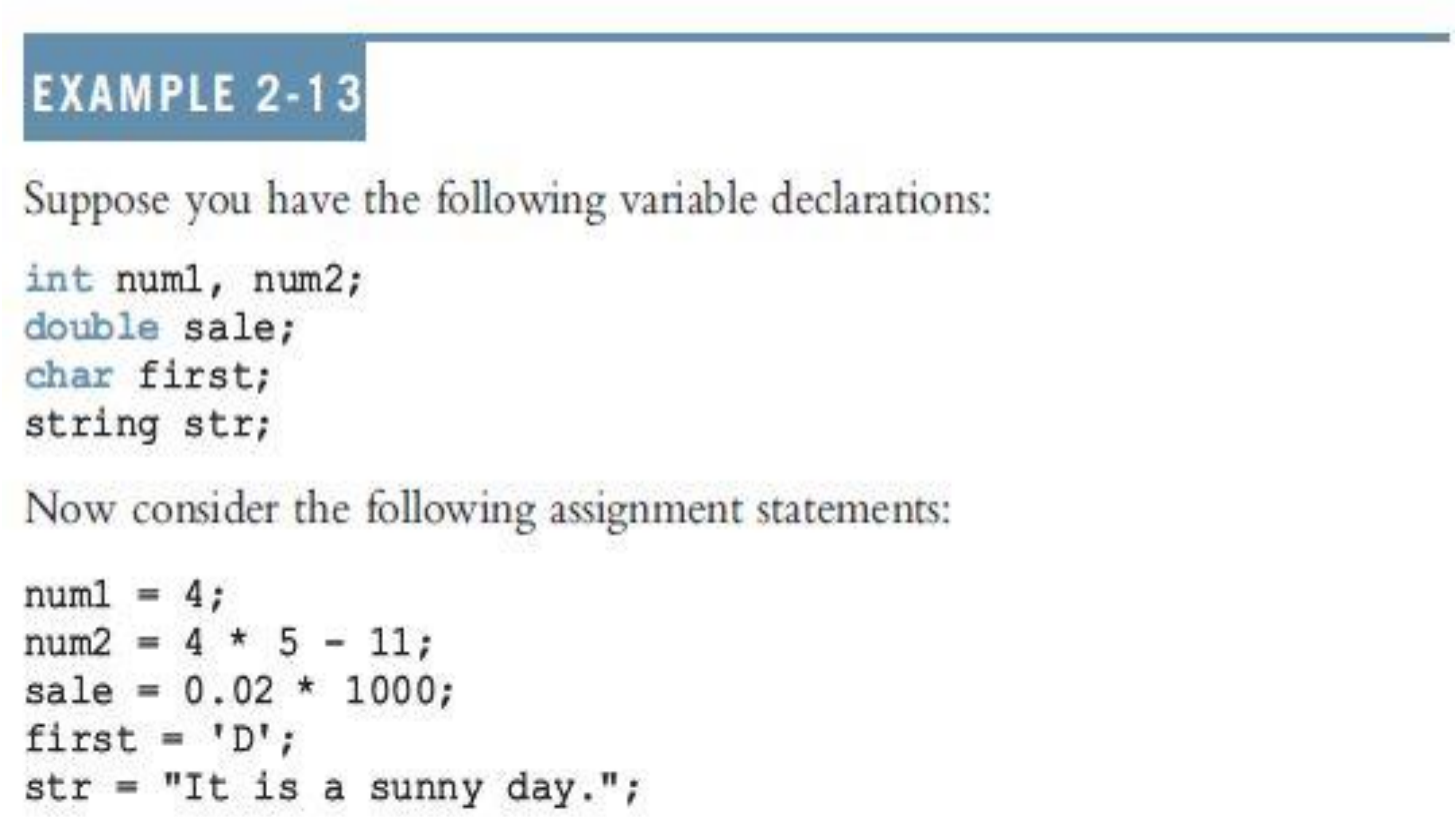
```
variable = expression;
```

Expression is evaluated and its value is assigned to the variable on the left side

A variable is said to be initialized the first time a value is placed into it

In C++, = is called the assignment operator

Assignment Statement (cont'd.)



EXAMPLE 2-13

Suppose you have the following variable declarations:

```
int num1, num2;
double sale;
char first;
string str;
```

Now consider the following assignment statements:

```
num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

Saving and Using the Value of an Expression

To save the value of an expression:

- Declare a variable of the appropriate data type
- Assign the value of the expression to the variable that was declared, use the assignment statement
- Wherever the value of the expression is needed, use the variable holding the value

Declaring & Initializing Variables

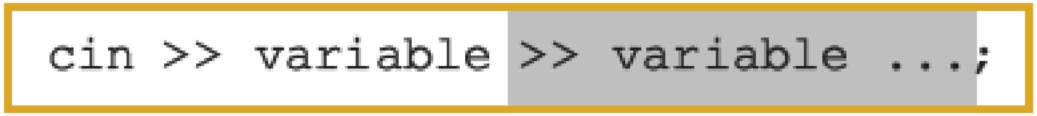
- C++ not automatically initialized variables.
- Variables can be initialized when declared:

```
int first=13, second=10;
char ch=' ';
double x=12.6;
```

- All variables must be initialized before they are used
  - But not necessarily during declaration

Input (Read) Statement

- `cin` is used with >> to gather input



```
cin >> variable >> variable ...;
```

- This is called an <u>input (read)</u> statement
- The <u>stream extraction operator</u> is >>
- For example, if miles is a double variable

```
cin >> miles;
```

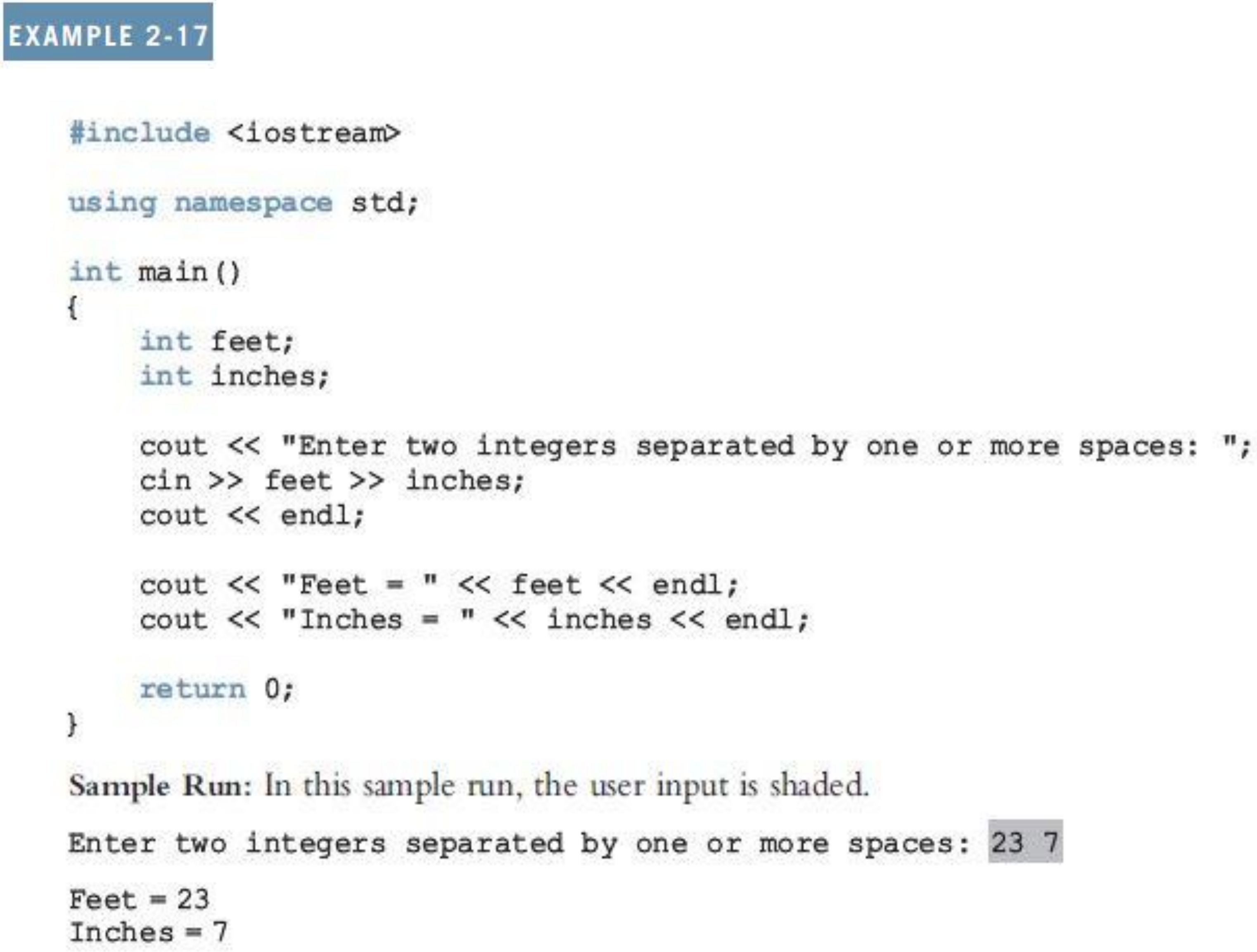
  - Causes computer to get a value of type `double` and places it in the variable `miles`

Input (Read) Statement (cont’d.)

- Using more than one variable in `cin` allows more than one value to be read at a time
- Example: if `feet` and `inches` are variables of type `int`, this statement:

```
cin >> feet >> inches;
```

  - Inputs two integers from the keyboard
  - Places them in variables `feet` and `inches` respectively



EXAMPLE 2-17

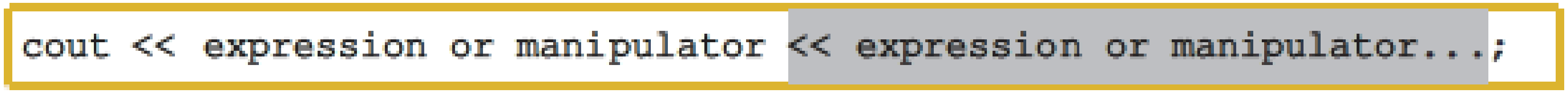
```
#include <iostream>

using namespace std;

int main()
{
    int feet;
    int inches;

    cout << "Enter two integers separated by one or more spaces: ";
    cin >> feet >> inches;
    cout << endl;

    cout << "Feet = " << feet << endl;
    cout << "Inches = " << inches << endl;

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
Enter two integers separated by one or more spaces: 23 7

Feet = 23
Inches = 7
```

Output

- The syntax of `cout` and << is:



```
cout << expression or manipulator << expression or manipulator...;
```

  - Called an <u>output statement</u>

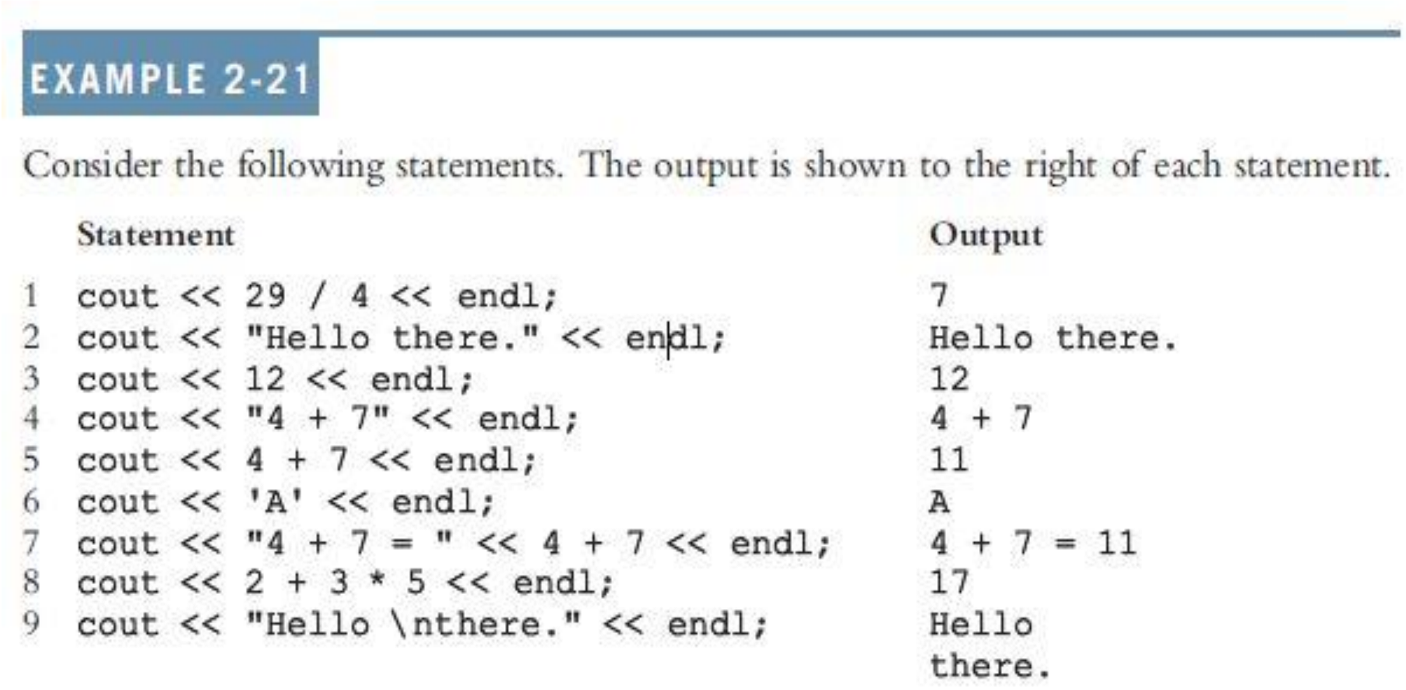
- The <u>stream insertion operator</u> is <<

- Expression evaluated and its value is printed at the current cursor position on the screen

Output (cont'd.)

A manipulator is used to format the output

- Example: `endl` causes insertion point to move to beginning of next line



EXAMPLE 2-21

Consider the following statements. The output is shown to the right of each statement.

```
   Statement                                   Output
1  cout << 29 / 4 << endl;                     7
2  cout << "Hello there." << endl;             Hello there.
3  cout << 12 << endl;                         12
4  cout << "4 + 7" << endl;                    4 + 7
5  cout << 4 + 7 << endl;                      11
6  cout << 'A' << endl;                        A
7  cout << "4 + 7 = " << 4 + 7 << endl;        4 + 7 = 11
8  cout << 2 + 3 * 5 << endl;                  17
9  cout << "Hello \nthere." << endl;           Hello
                                               there.
```

Output (cont’d.)

- The new line character is '\n’ (called an escape sequence
  - May appear anywhere in the string

```
cout << "Hello there.";
cout << "My name is James.";
```

Output:

```
Hello there.My name is James.
```

```
cout << "Hello there.\n";
cout << "My name is James.";
```

Output :

```
Hello there.
My name is James.
```

Output (cont’d.)

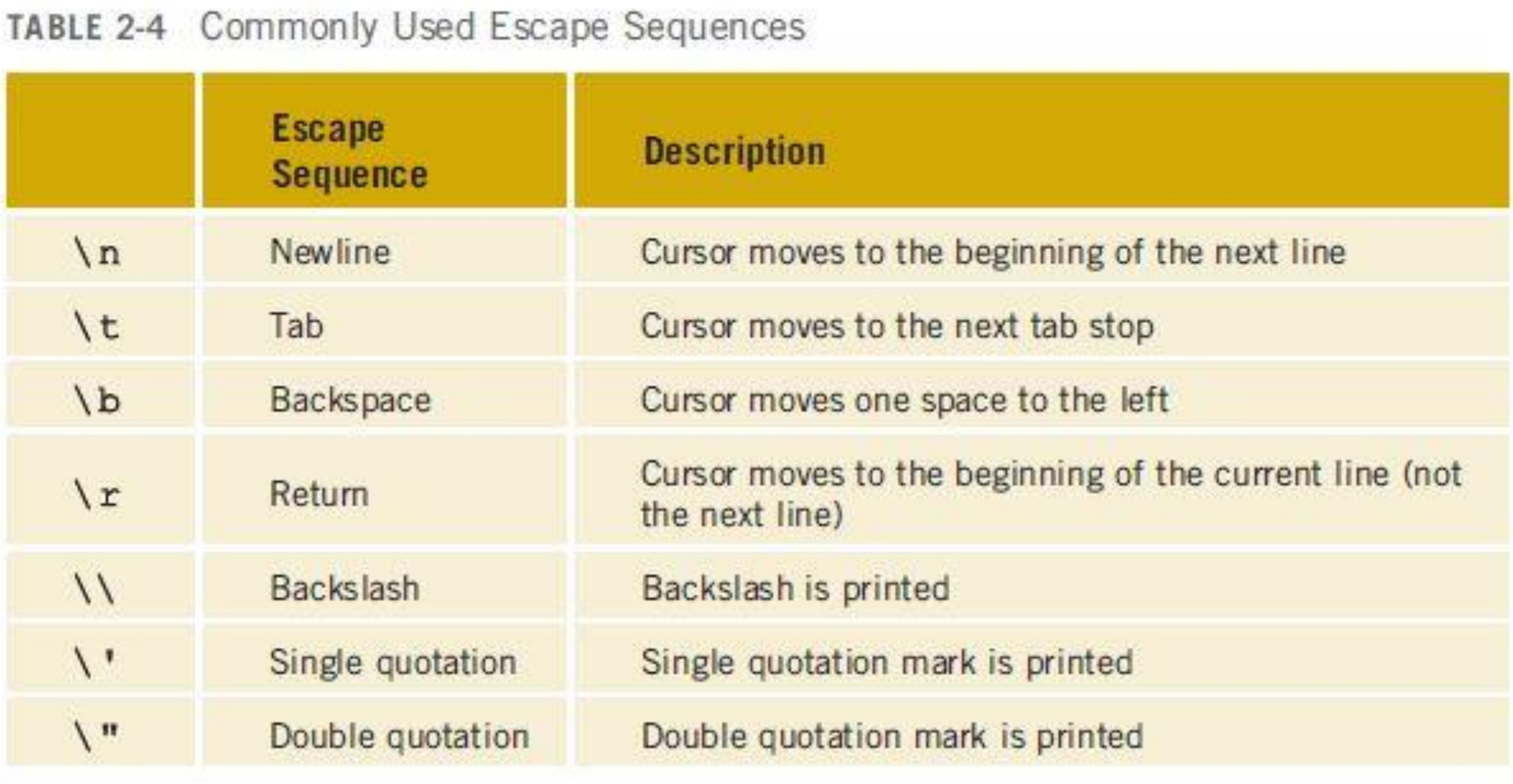


TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Increment and Decrement Operators

- Increment operator: increase variable by 1
  - Pre-increment: `++variable`
  - Post-increment: `variable++`
- Decrement operator: decrease variable by 1
  - Pre-decrement: `--variable`
  - Post-decrement: `variable–`
- What is the difference between the following?

```
x = 5;
y = ++x;
```

```
x = 5;
y = x++;
```

More on Assignment Statements

- Two forms of assignment
  - Simple and compound
- Simple assignment:
  - x=10; x=y+10;
- Compound assignment operators provide more concise notation
  - They provide a shorthand form of simple assignment of the form: `x = x * y;`
- Compound assignment:

```
x *= y;
```

More on Assignment Statements

- Compound operators are available for all arithmetic operators, i.e. `+=, -=, *=, /=, %=`
- General form of using compound assignment operator is as follows:

```
variable [operator]= expression;
```

  - Where [operator] can be any arithmetic operator.
  - The above compound assignment is equal to the following simple assignment.
    - `variable = variable [operator] expression;`
- Examples: x+=y; x+=(y*2); x*=(y+z/2); x/=2;

Preprocessor Directives

- C++ has a small number of operations
- Many functions and symbols needed to run a C++ program are provided as collection of libraries
- Every library has a name and is referred to by a header file
- Preprocessor directives are commands supplied to the preprocessor program
- All preprocessor commands begin with #
- No semicolon at the end of these commands

Preprocessor Directives (cont'd.)

- Syntax to include a header file:

```
#include <headerFileName>
```

- For example:

```
#include <iostream>
```

  - Causes the preprocessor to include the header file `iostream` in the program

- Preprocessor commands are processed before the program goes through the compiler

namespace and Using cin and cout in a Program

- cin and cout are declared in the header file iostream, but within std namespace
- To use cin and cout in a program, use the following two statements:

```
#include <iostream>

using namespace std;
```

Using the string Data Type in a Program

- To use the string type, you need to access its definition from the header file string
- Include the following preprocessor directive:

```
#include  <string>
```

Creating a C++ Program

- A C++ program is a collection of functions, one of which is the function `main`
- The first line of the function `main` is called the heading of the function:
  - `int main()`
- The statements enclosed between the curly braces ({ and }) form the body of the function

Creating a C++ Program (cont’d.)

- A C++ program contains two types of statements:
  - Declaration statements: declare things, such as variables
  - Executable statements: perform calculations, manipulate data, create output, accept input, etc.

Debugging: Understanding and Fixing Syntax Errors

- Compile a program
  - Compiler will identify the syntax errors
  - Specifies the line numbers where the errors occur

```
Example2_Syntax_Errors.cpp
c:\chapter 2 source code\example2_syntax_errors.cpp(9) :
  error C2146: syntax error :
missing ';' before identifier 'num'
c:\chapter 2 source code\example2_syntax_errors.cpp(11) :
  error C2065: 'tempNum' :
undeclared identifier
```

Program Style and Form: Syntax

- Syntax rules: indicate what is legal and what is not legal
- Errors in syntax are found in compilation
- Sometimes, the line number which compiler reports does not contain the actual error.
  - In this case, the error may be in some other line near the line, which compiler reported.

```
int x;                //Line 1
int y                 //Line 2: error
double z;  //Line 3
t = w + x; //Line 4: error
```

Use of Semicolons, Brackets, and Commas

- All C++ statements end with a semicolon
  - Also called a statement terminator
- { and } are not C++ statements
  - Are used grouping of statements
- Commas separate items in a list

Semantics

Semantics: set of rules that gives meaning to a language

- Possible to remove all syntax errors in a program and still not have it run successfully
  - it may not do what you meant it to do

Ex: `2 + 3 * 5` and `(2 + 3) * 5`

are both syntactically correct expressions, but have different meanings

Naming Identifiers

Identifiers can be self-documenting:

- `CENTIMETERS_PER_INCH`

Avoid run-together words :

- `annualsale`
- Solution:
  - Capitalizing the beginning of each new word: `annualSale`
  - Inserting an underscore just before a new word: `annual_sale`

Prompt Lines

- Prompt lines: executable statements that inform the user what to do

```
cout << "Please enter a number between 1 and 10
 and " << "press the return key" << endl;

cin >> num;
```

- Always include prompt lines when input is needed from users

Documentation

- A well-documented program is easier to understand and modify
- You use comments to document programs
- Comments should appear in a program to:
  - Explain the purpose of the program
  - Identify who wrote it
  - Explain the purpose of particular statements

Form and Style

- Consider two ways of declaring variables:
  - Method 1

```
int feet, inch;
double x, y;
```

  - Method 2

```
int feet,inch; double x,y;
```

- Both are correct; however, the second is hard to read

Program Indentation

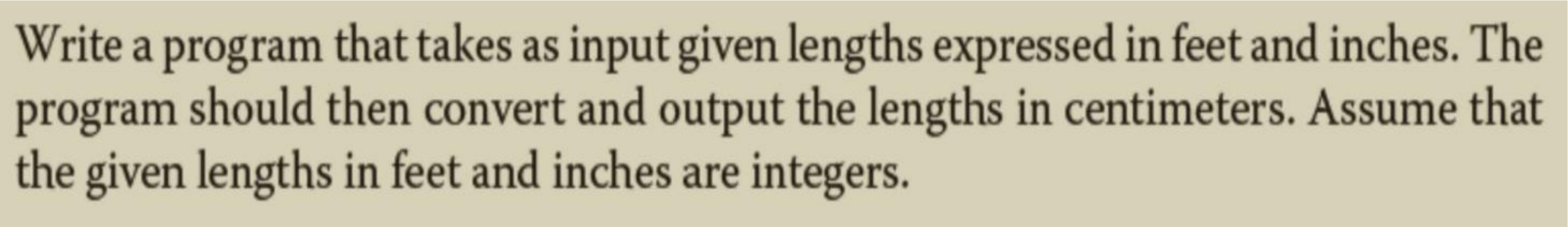
- Indentation refers to whitespaces in the beginning of statements.
  - Indentation improves program readability.

```
int main()
{
cout<<“Bismillah”;
return 0;
}
```

```
int main()
{
    cout<<“Bismillah”;
    return 0;
}
```

- Both programs are correct, but the second one is more readable and beautified code.

Practice Problem 1

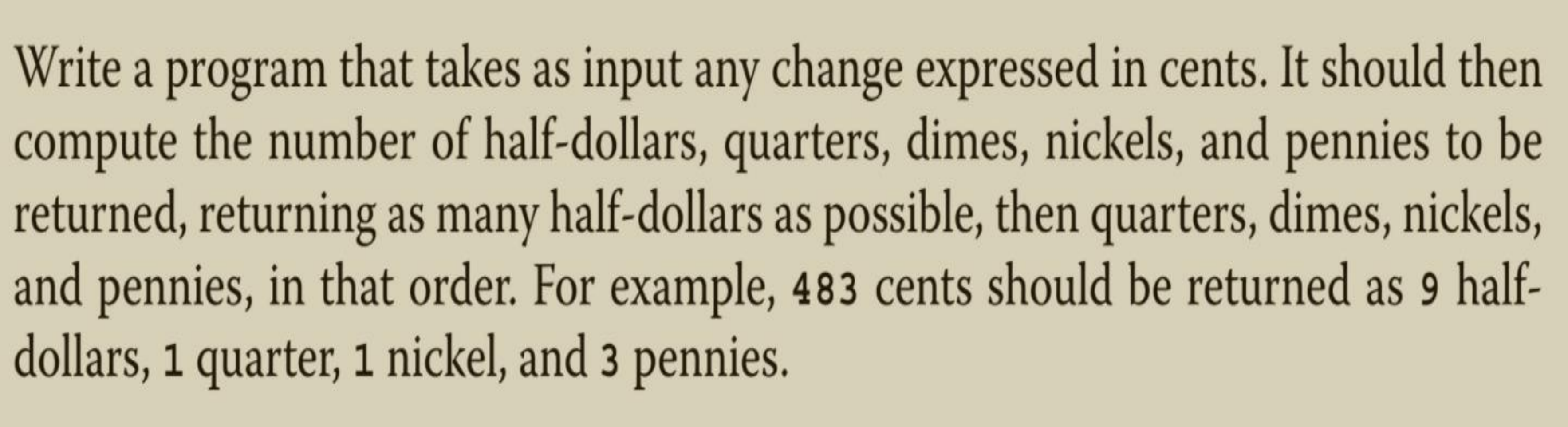


Write a program that takes as input given lengths expressed in feet and inches. The program should then convert and output the lengths in centimeters. Assume that the given lengths in feet and inches are integers.

Hints:

1 inch =2.54 centimeter
1 foot =12 inches

Practice Problem 2



Write a program that takes as input any change expressed in cents. It should then compute the number of half-dollars, quarters, dimes, nickels, and pennies to be returned, returning as many half-dollars as possible, then quarters, dimes, nickels, and pennies, in that order. For example, `483` cents should be returned as `9` half-dollars, `1` quarter, `1` nickel, and `3` pennies.

Hints:

1 half dollar =50
1 quarter = 25
1 dimes = 10
1 nickle = 5

Summary

- C++ program: collection of functions, one of which is always called `main`
- Identifiers consist of letters, digits, and underscores, and begins with letter or underscore
- The arithmetic operators in C++ are addition (+), subtraction (-), multiplication (*), division (/), and modulus (%)
- Arithmetic expressions are evaluated using the precedence associativity rules

Summary (cont’d.)

- All operands in an integral expression are integers
- All operands in a floating-point expression are decimal numbers
- Mixed expression: contains both integers and decimal numbers
- Use the cast operator to explicitly convert values from one data type to another
- A named constant is initialized when declared
- All variables must be declared before used

Summary (cont'd.)

- Use `cin` and stream extraction operator >> to input from the standard input device
- Use `cout` and stream insertion operator << to output to the standard output device
- Preprocessor commands are processed before the program goes through the compiler
- A file containing a C++ program usually ends with the extension `.cpp`