

Chapter 2:

Introduction to C++



Addison-Wesley
is an imprint of

PEARSON

Copyright © 2012 Pearson Education, Inc.

2.1



The Parts of a C++ Program

The Parts of a C++ Program

```
// sample C++ program    comment
#include <iostream>
using namespace std;      which namespace to use

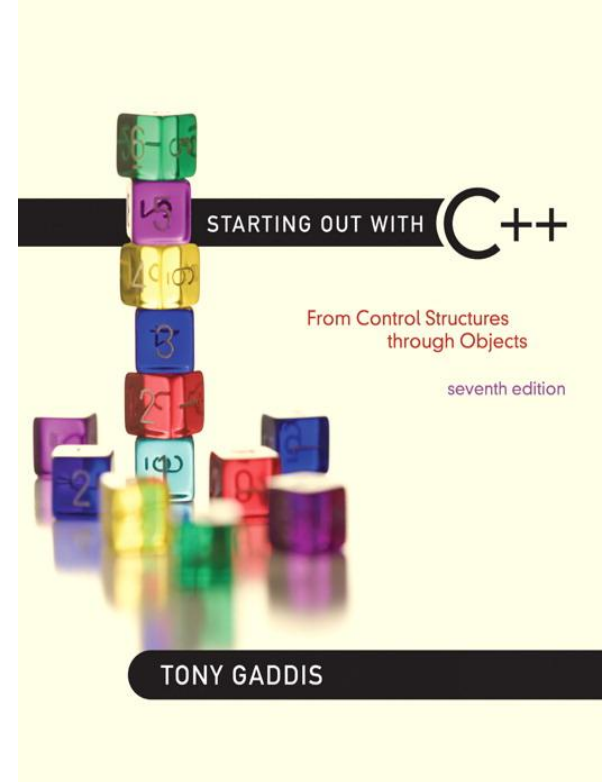
int main()    beginning of function named main
{    beginning of block for main
    cout << "Hello, there!";    string literal    output statement
    return 0;    send 0 to operating system
}    end of block for main
```

Special Characters

Character	Name	Meaning
//	Double slash	Beginning of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
()	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

2.2

The `cout` Object



The cout Object

- Displays output on the computer screen
- You use the stream insertion operator << to send output to `cout`:

```
cout << "Programming is fun!";
```

The cout Object

- Can be used to send more than one item to `cout`:

```
cout << "Hello " << "there!";
```

Or:

```
cout << "Hello ";  
cout << "there!";
```

The cout Object

- This produces one line of output:

```
cout << "Programming is ";  
cout << "fun!";
```


The endl Manipulator

- You can use the **endl** manipulator to start a new line of output. This will produce two lines of output:

```
cout << "Programming is" << endl;  
cout << "fun!";
```

The endl Manipulator

```
cout << "Programming is" << endl;  
cout << "fun!";
```



The endl Manipulator

- You do NOT put quotation marks around **endl**
- The last character in **endl** is a lowercase L, not the number 1.

endl ← This is a lowercase L

The `\n` Escape Sequence

- You can also use the `\n` escape sequence to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";  
cout << "fun!";
```

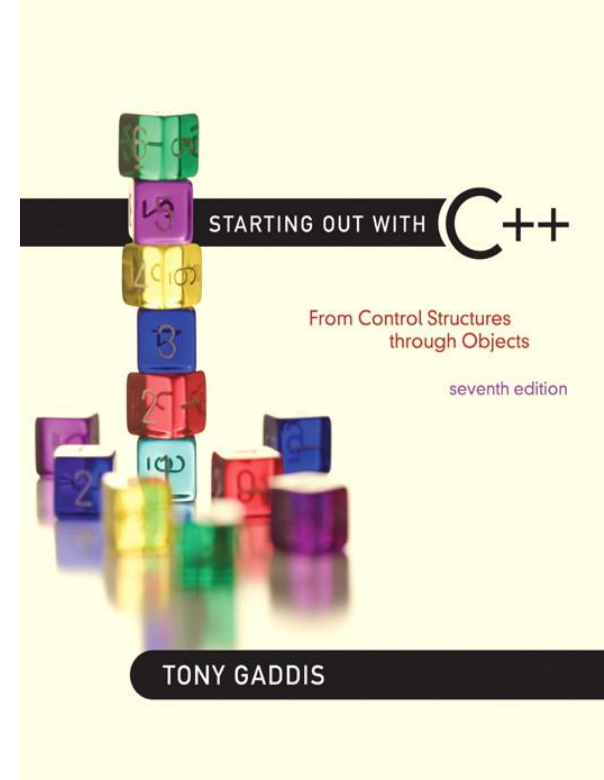
Notice that the `\n` is INSIDE
the string.

The `\n` Escape Sequence

```
cout << "Programming is\n";  
cout << "fun!";
```



2.3



The `#include` Directive

The #include Directive

- Inserts the contents of another file into the program
- This is a **preprocessor directive**, not part of C++ language
- **#include** lines not seen by compiler
- Do not place a semicolon at end of **#include** line

2.4



Variables and Literals

Variables and Literals


- Variable:
 - a storage location in memory
 - It has a name and a type of data it can hold
 - It must be defined before it can be used:

```
int item;
```

Variable Definition in Program 2-7

Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int number;
8
9      number = 5;
10     cout << "The value in number is " << number << endl;
11     return 0;
12 }
```



Program Output

The value in number is 5

Literal

- Literal:
 - a value that is written into a program's code.

"hello, there" (string literal)

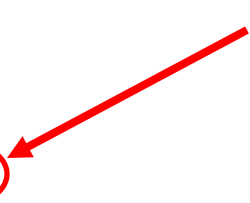
12 (integer literal)

Integer Literal in Program 2-9

Program 2-9

```
1 // This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

20 is an integer literal



Program Output

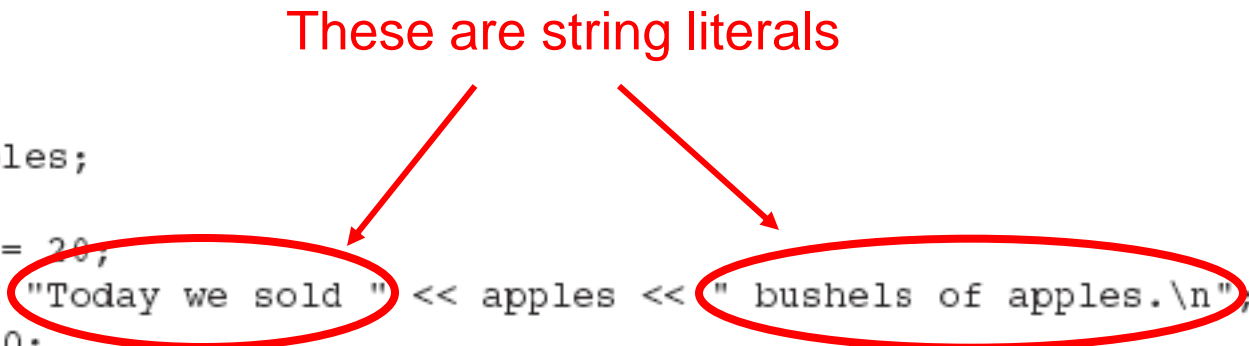
Today we sold 20 bushels of apples.

String Literals in Program 2-9

Program 2-9

```
1  // This program has literals and a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int apples;
8
9      apples = 20;
10     cout << "Today we sold " << apples << " bushels of apples.\n";
11     return 0;
12 }
```

These are string literals

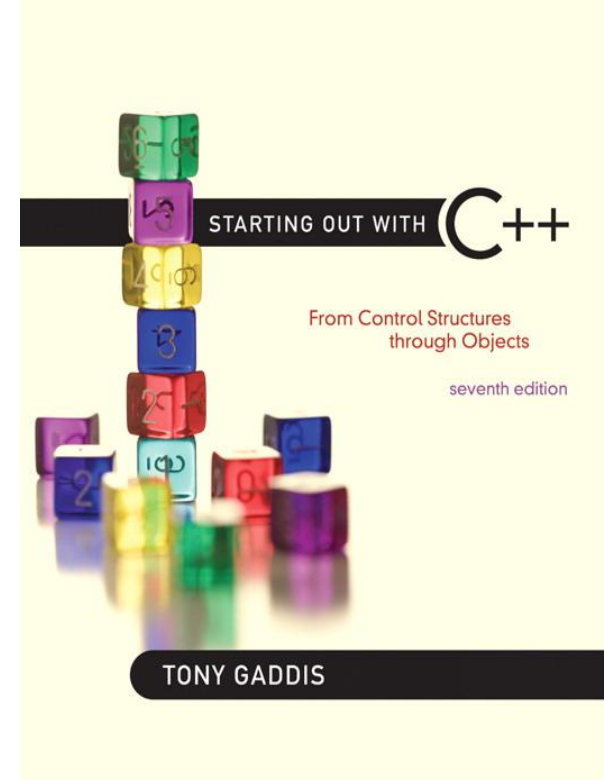


Program Output

Today we sold 20 bushels of apples.

2.5

Identifiers



Identifiers

- An identifier is a **programmer-defined name** for some element in a program:
 - variable
 - function
 - named constant

C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

Table 2-4 The C++ Key Words

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Variable Names

- A variable name should represent the purpose of the variable. For example:

itemsOrdered

The purpose of this variable is to hold the number of items ordered.

Identifier Rules

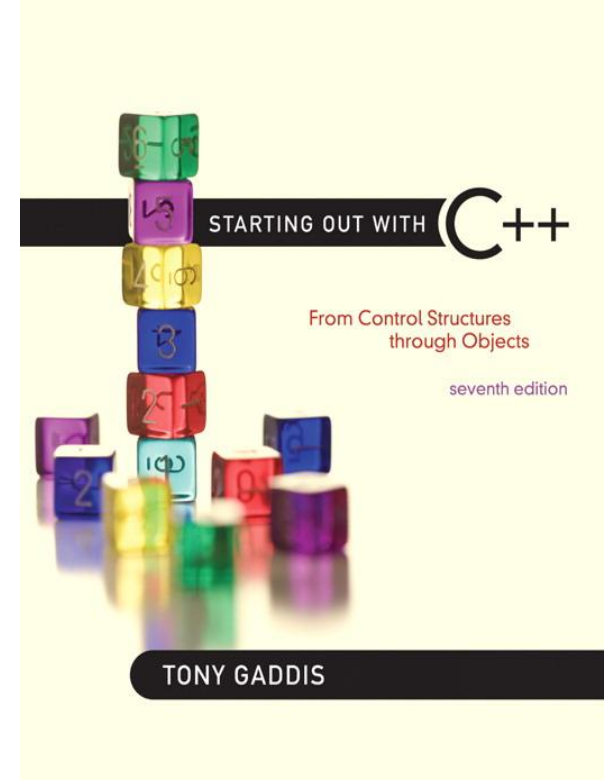
- The first character of an identifier must be an alphabetic character or an underscore (_),
- After the first character you may use alphabetic characters, numbers, or underscore characters.
- Upper- and lowercase characters are distinct

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
totalSales	Yes	
total_Sales	Yes	
total.Sales	No	Cannot contain .
4thQtrSales	No	Cannot begin with digit
totalSale\$	No	Cannot contain \$

2.6

Integer Data Types



Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

Table 2-6 Integer Data Types, Sizes, and Ranges

Data Type	Size	Range
short	2 bytes	-32,768 to +32,767
unsigned short	2 bytes	0 to +65,535
int	4 bytes	-2,147,483,648 to +2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
long	4 bytes	-2,147,483,648 to +2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Defining Variables

- Variables of the same type can be defined
 - On separate lines:

```
int length;  
int width;  
unsigned int area;
```
 - On the same line:

```
int length, width;  
unsigned int area;
```
- Variables of different types must be in different definitions

Integer Types in Program 2-10

Program 2-10

```
1  // This program has variables of several of the integer types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int checking;
8      unsigned int miles;
9      long days;
10
11     checking = -20;
12     miles = 4276;
13     days = 189000;
14     cout << "We have made a long journey of " << miles;
15     cout << " miles.\n";
16     cout << "Our checking account balance is " << checking;
17     cout << "\nAbout " << days << " days ago Columbus ";
18     cout << "stood on this spot.\n";
19     return 0;
20 }
```

This program has three variables: checking, miles, and days

Integer Literals

- An integer literal is an integer value that is typed into a program's code. For example:

```
itemsOrdered = 15;
```

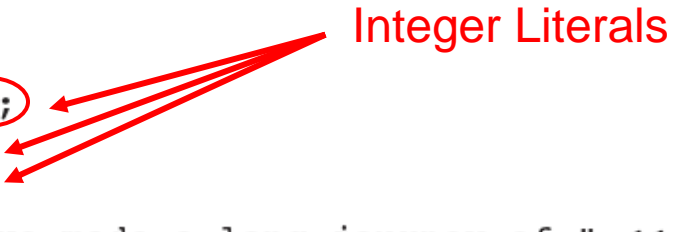
In this code, **15** is an integer literal.

Integer Literals in Program 2-10

Program 2-10

```
1  // This program has variables of several of the integer types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int checking;
8      unsigned int miles;
9      long days;
10
11     checking = -20;
12     miles = 4276;
13     days = 189000;
14     cout << "We have made a long journey of " << miles;
15     cout << " miles.\n";
16     cout << "Our checking account balance is " << checking;
17     cout << "\nAbout " << days << " days ago Columbus ";
18     cout << "stood on this spot.\n";
19     return 0;
20 }
```

Integer Literals



Integer Literals

- Integer literals are stored in memory as type `int` by default
- To store an integer constant in a long memory location, put '`L`' at the end of the number: `1234L`
- Constants that begin with '`0`' (zero) are base 8: `075`
- Constants that begin with '`0x`' are base 16: `0x75A`

2.7



The `char` Data Type

The char Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:
`char letter;`
`letter = 'C';`

MEMORY:
letter



67

Character Literals

- Character literals must be enclosed in single quote marks.

Example: 'A'

Character Literals in Program 2-13

Program 2-13

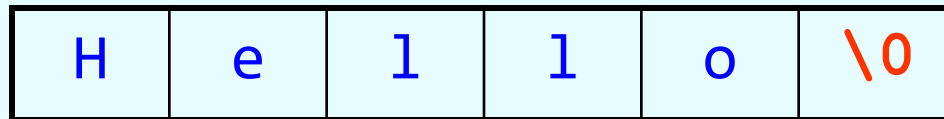
```
1  // This program uses character literals.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      char letter;
8
9      letter = 'A';
10     cout << letter << endl;
11     letter = 'B';
12     cout << letter << endl;
13     return 0;
14 }
```

Program Output

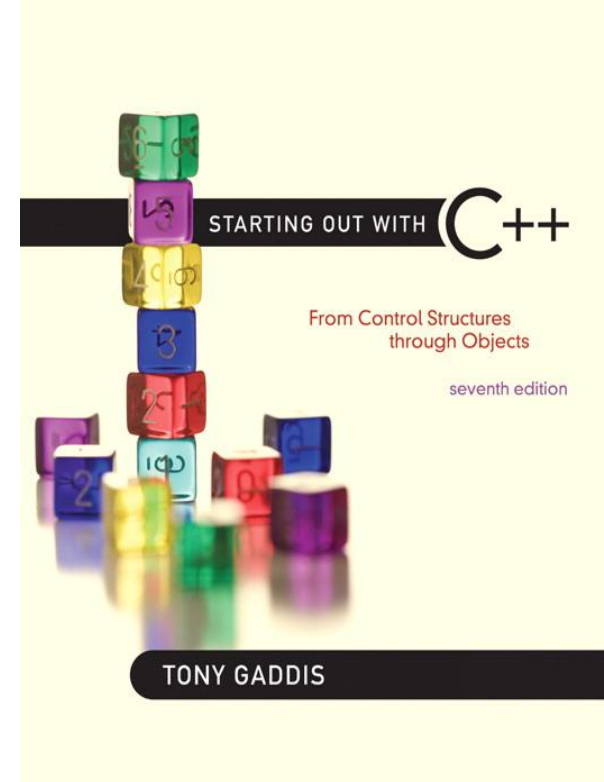
A
B

Character Strings [c-strings]

- A series of characters in consecutive memory locations: "Hello"
- Stored with the null terminator, \0, at the end:
- Comprised of the characters between the " "



2.8



The C++ `string` Class

The C++ string Class

- Special data type supports working with strings

```
#include <string>
```

- Can define `string` variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "George";
```

```
lastName  = "Washington";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

The string class in Program 2-15

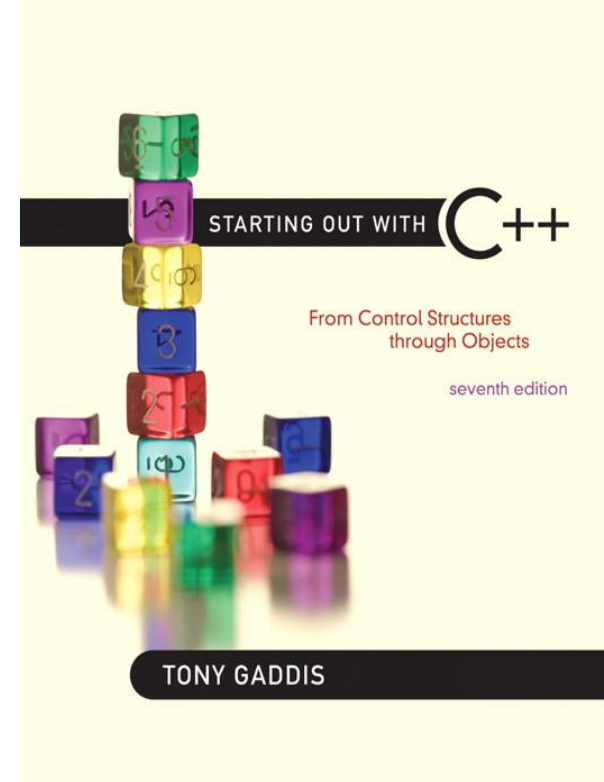
Program 2-15

```
1  // This program demonstrates the string class.
2  #include <iostream>
3  #include <string> // Required for the string class.
4  using namespace std;
5
6  int main()
7  {
8      string movieTitle;
9
10     movieTitle = "Wheels of Fury";
11     cout << "My favorite movie is " << movieTitle << endl;
12     return 0;
13 }
```

Program Output

My favorite movie is Wheels of Fury

2.9



Floating-Point Data Types

Floating-Point Data Types

- The floating-point data types are:

`float`
`double`
`long double`

- They can hold real numbers such as:

`12.45` `-3.8`

- Stored in a form similar to **scientific notation**
- All floating-point numbers are signed

Floating-Point Data Types

Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	float	4 bytes. Numbers between $\pm 3.4\text{E-}38$ and $\pm 3.4\text{E}38$
Double precision	double	8 bytes. Numbers between $\pm 1.7\text{E-}308$ and $\pm 1.7\text{E}308$
Long double precision	long double*	8 bytes. Numbers between $\pm 1.7\text{E-}308$ and $\pm 1.7\text{E}308$

Floating-Point Literals

- Can be represented in
 - Fixed point (decimal) notation:
31.4159 0.0000625
 - E notation:
3.14159E1 6.25e-5
- Are **double** by default
- Can be forced to be **float** (3.14159**f**) or long double (0.0000625**L**)

Floating-Point Data Types in Program 2-16

Program 2-16

```
1  // This program uses floating point data types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      float distance;
8      double mass;
9
10     distance = 1.495979E11;
11     mass = 1.989E30;
12     cout << "The Sun is " << distance << " meters away.\n";
13     cout << "The Sun\'s mass is " << mass << " kilograms.\n";
14     return 0;
15 }
```

Program Output

The Sun is 1.49598e+011 meters away.
The Sun's mass is 1.989e+030 kilograms.

2.10



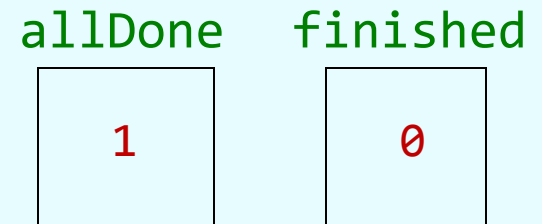
The `bool` Data Type

The bool Data Type

- Represents values that are **true** or **false**
- **bool** variables are stored as small integers
- **false** is represented by **0**, **true** by **1**:

```
bool allDone = true;
```

```
bool finished = false;
```



Boolean Variables in Program 2-17

Program 2-17

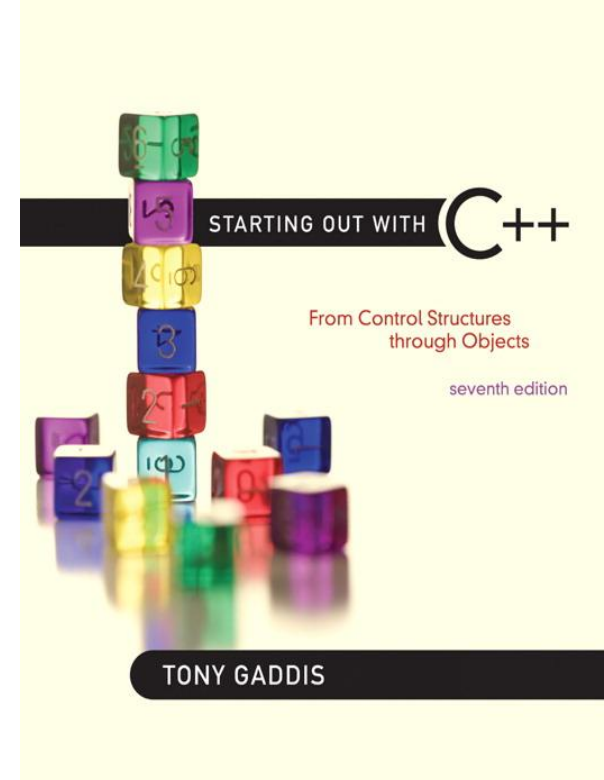
```
1  // This program demonstrates boolean variables.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      bool boolValue;
8
9      boolValue = true;
10     cout << boolValue << endl;
11     boolValue = false;
12     cout << boolValue << endl;
13     return 0;
14 }
```

Program Output

1
0

2.11

Determining the Size of a Data Type



Determining the Size of a Data Type

The `sizeof` operator gives the size of any data type or variable:

```
double amount;
```

```
cout << "A double is stored in "  
      << sizeof(double) << "bytes\n";
```

```
cout << "Variable amount is stored in "  
      << sizeof(amount)  
      << "bytes\n";
```

2.12



Variable Assignments and Initialization

Variable Assignments and Initialization

- An assignment statement uses the = operator to store a value in a variable.

```
item = 12;
```

- This statement assigns the value 12 to the `item` variable.

Assignment

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

```
// ERROR!  
12 = item;
```

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```


Variable Initialization in Program 2-19

Program 2-19

```
1  // This program shows variable initialization.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int month = 2, days = 28;
8
9      cout << "Month " << month << " has " << days << " days.\n";
10     return 0;
11 }
```

Program Output

Month 2 has 28 days.

2.13

Scope



Scope

- The scope of a variable:
 - the part of the program in which the variable can be accessed
- A variable cannot be used before it is defined

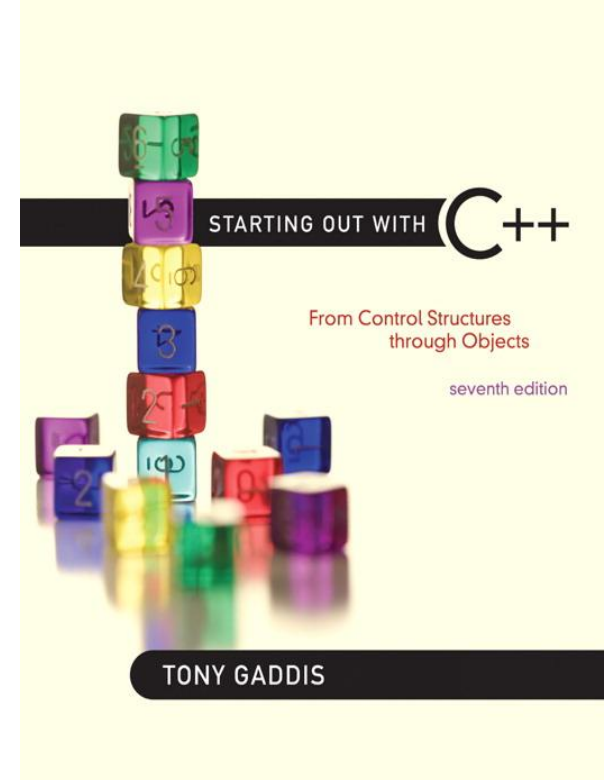
Variable Out of Scope in Program 2-20

Program 2-20

```
1  // This program can't find its variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      cout << value; // ERROR! value not defined yet!
8
9      int value = 100;
10     return 0;
11 }
```

2.14

Arithmetic Operators



Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
 - unary (1 operand) -5
 - binary (2 operands) $13 - 7$
 - ternary (3 operands) $\text{exp1} ? \text{exp2} : \text{exp3}$

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	<code>ans = 7 + 3;</code>	10
-	subtraction	<code>ans = 7 - 3;</code>	4
*	multiplication	<code>ans = 7 * 3;</code>	21
/	division	<code>ans = 7 / 3;</code>	2
%	modulus	<code>ans = 7 % 3;</code>	1

Arithmetic Operators in Program 2-21

Program 2-21

```
1  // This program calculates hourly wages, including overtime.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      double regularWages,           // To hold regular wages
8          basePayRate = 18.25,       // Base pay rate
9          regularHours = 40.0,       // Hours worked less overtime
10         overtimeWages,             // To hold overtime wages
11         overtimePayRate = 27.78,    // overtime pay rate
12         overtimeHours = 10,         // Overtime hours worked
13         totalWages;                // To hold total wages
14
15     // Calculate the regular wages.
16     regularWages = basePayRate * regularHours;
17
18     // Calculate the overtime wages.
19     overtimeWages = overtimePayRate * overtimeHours;
20
21     // Calculate the total wages.
22     totalWages = regularWages + overtimeWages;
23
24     // Display the total wages.
25     cout << "Wages for this week are $" << totalWages << endl;
26     return 0;
27 }
```

Program Output

Wages for this week are \$1007.8

A Closer Look at the / Operator

- / (division) operator performs **integer** division if both operands are integers

```
cout << 13 / 5;    // displays 2  
cout << 91 / 7;    // displays 13
```

- If either operand is **floating-point**, the result is floating-point

```
cout << 13 / 5.0;  // displays 2.6  
cout << 91.0 / 7;  // displays 13.0
```

A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division

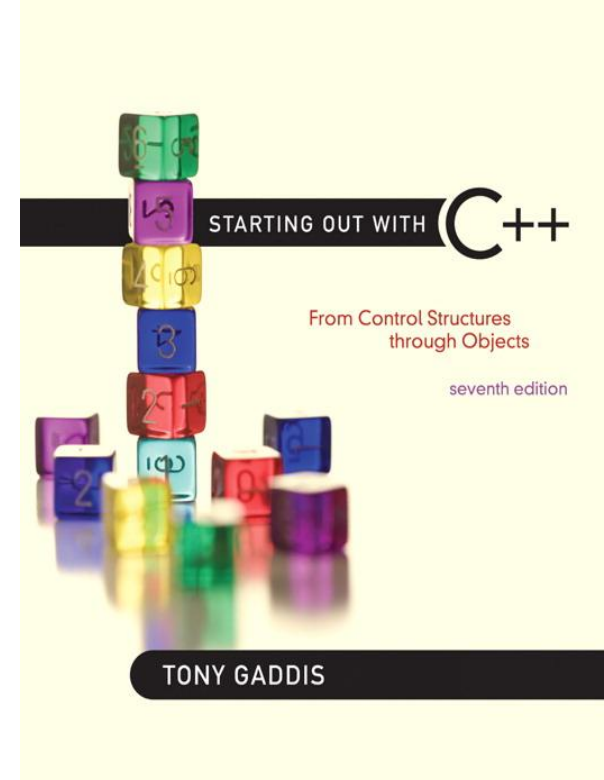
```
cout << 13 % 5;    //displays 3
```

- % requires integers for both operands

```
cout << 13 % 5.0; //error
```

2.15

Comments



Comments

- Used to document parts of the program
- Intended for persons reading the source code of the program:
 - Indicate the purpose of the program
 - Describe the use of variables
 - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

Begin with `//` through to the end of line:

```
int length = 12; // length in inches  
int width = 15;  // width in inches  
int area;        // calculated area
```

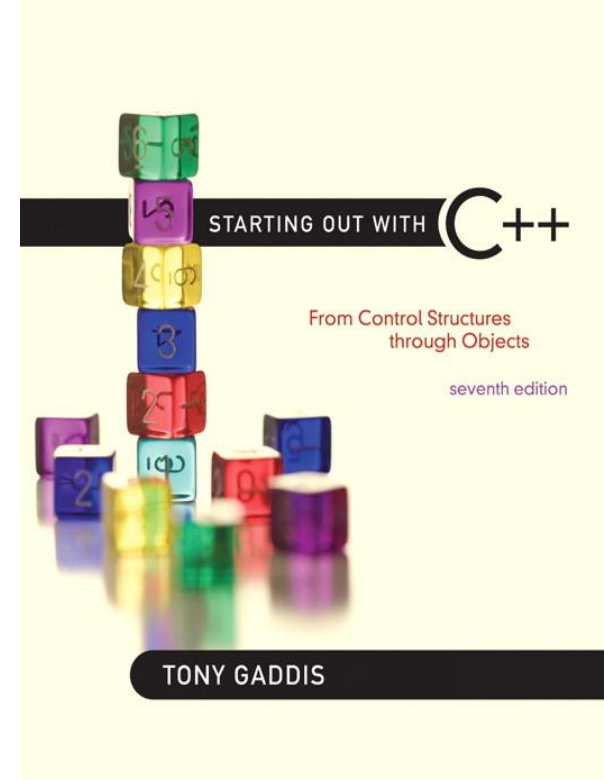
```
// calculate rectangle area  
area = length * width;
```

Multi-Line Comments

- Begin with `/*`, end with `*/`
- Can span multiple lines:
`/* this is a multi-line
comment
*/`
- Can begin and end on the same line:
`int area; /* calculated area */`

2.16

Named Constants



Named Constants

- Named constant (constant variable):
 - variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:


```
const double TAX_RATE = 0.0675;  
const int NUM_STATES = 50;
```
- Often named in uppercase letters

Named Constants in Program 2-28

Program 2-28

```
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // Constants
8     const double PI = 3.14159;
9     const double DIAMETER = 10.0;
10
11     // Variable to hold the circumference
12     double circumference;
13
14     // Calculate the circumference.
15     circumference = PI * DIAMETER;
16
17     // Display the circumference.
18     cout << "The circumference is: " << circumference << endl;
19     return 0;
20 }
```

Program Output

The circumference is: 31.4159

2.17

Programming Style



Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Does not affect the syntax of the program
- Affects the readability of the source code

Programming Style

Common elements to improve readability:

- Braces { } aligned vertically
- Indentation of statements within a set of braces
- Blank lines between declaration and other statements
- Long statements wrapped over multiple lines with aligned operators

2.18



Standard and Prestandard C++

Standard and Prestandard C++

Older-style C++ programs:

- Use `.h` at end of header files:
- `#include <iostream.h>`
- Use `#define` preprocessor directive instead of `const` definitions
- Do not use `using namespace` convention
- May not compile with a standard C++ compiler

#define directive in Program 2-31

Program 2-31

```
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 #define PI 3.14159
6 #define DIAMETER 10.0
7
8 int main()
9 {
10     // Variable to hold the circumference
11     double circumference;
12
13     // Calculate the circumference.
14     circumference = PI * DIAMETER;
15
16     // Display the circumference.
17     cout << "The circumference is: " << circumference << endl;
18     return 0;
19 }
```

Program Output

The circumference is: 31.4159