An Introduction to Programming with C++

Fifth Edition

Chapter 4

Chapter 4: Variables, Constants, and Arithmetic Operators

Objectives

- Distinguish among a variable, a named constant, and a literal constant
- Explain how data is stored in memory
- Declare and initialize a memory location
- Use an assignment statement to assign data to a variable

Objectives (continued)

- Include arithmetic operators and arithmetic assignment operators in a statement
- Get string input using the `getline()` function
- Ignore characters using the `ignore()` function
- Format floating-point output
- Write simple .NET C++ commands

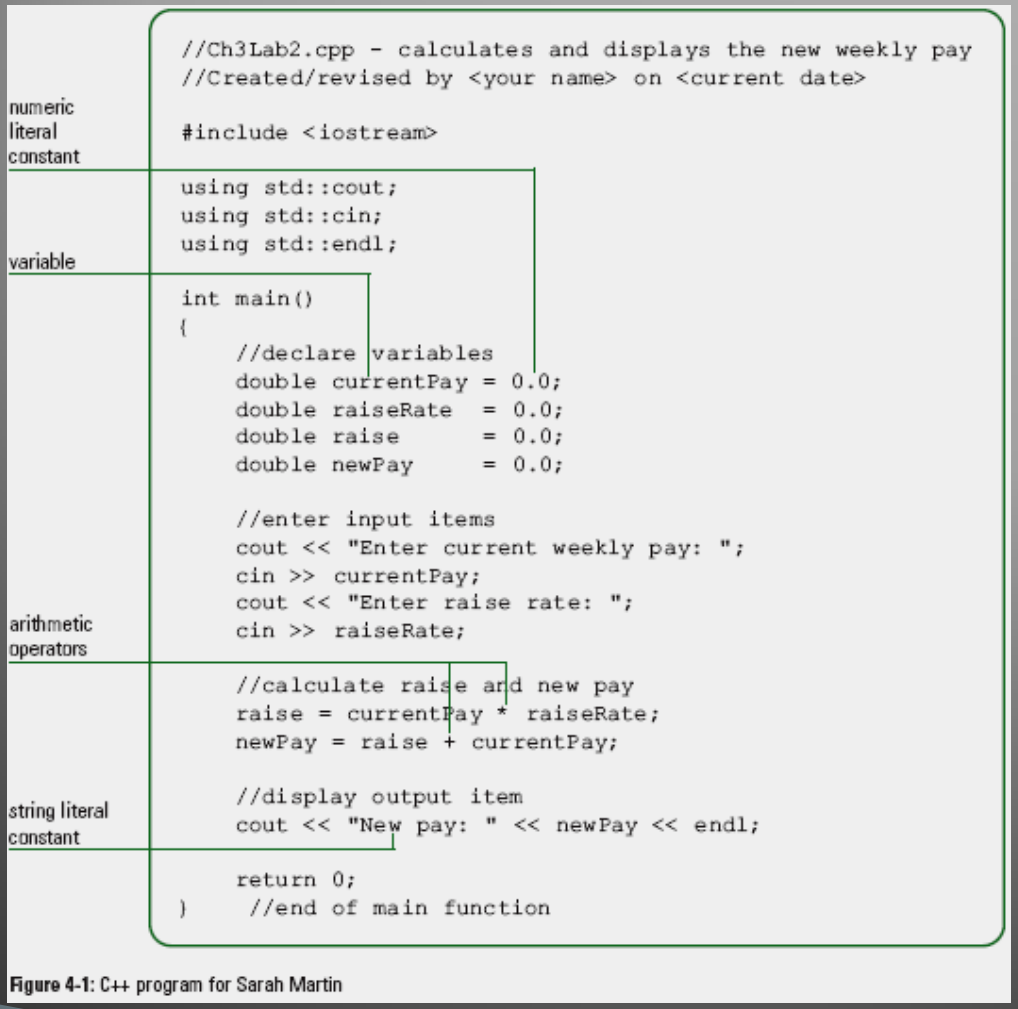
Concept Lesson

- More on the Problem-Solving Process
- Variables and Named Constants
- Declaring a Named Constant
- Declaring a Variable

Concept Lesson (continued)

- Using an Assignment Statement to Store Data in a Variable
- Arithmetic Operators
- Getting Data from the Keyboard
- Formatting Floating-Point Numbers

More on the Problem-Solving Process




```
//Ch3Lab2.cpp - calculates and displays the new weekly pay
//Created/revised by <your name> on <current date>

#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
     //declare variables
     double currentPay = 0.0;
     double raiseRate  = 0.0;
     double raise      = 0.0;
     double newPay     = 0.0;

     //enter input items
     cout << "Enter current weekly pay: ";
     cin >> currentPay;
     cout << "Enter raise rate: ";
     cin >> raiseRate;

     //calculate raise and new pay
     raise = currentPay * raiseRate;
     newPay = raise + currentPay;

     //display output item
     cout << "New pay: " << newPay << endl;

     return 0;
}    //end of main function
```

Figure 4-1: C++ program for Sarah Martin

Variables and Named Constants

- Declare a memory location for each input, processing, and output item in IPO chart
  - A **variable** is a type of memory location whose contents can change while program is running
  - Values of **named constant** items remain the same each time the program is executed

Variables and Named Constants (continued)

Problem specification:

Mr. Johnson needs a program that calculates and displays the area of a circle, which is based on the circle radius he enters. The formula for calculating the area of a circle is Πr^2, where Π represents pi and r represents the radius. You will use 3.141593 as the value for pi.

Input	Processing	Output
radius pi (3.141593)	Processing items: radius squared Algorithm: 1. enter the radius 2. calculate the radius squared by multiplying the radius by itself 3. calculate the area by multiplying pi by the radius squared 4. display the area	area

Figure 4-2: Problem specification and IPO chart for the circle area problem

Requires four memory locations:

- Two input items
  - radius
    - variable
  - pi
    - named constant
- One processing item
  - radius squared
    - variable
- One output item
  - area
    - variable

Selecting a Name for a Memory Location

- Identifiers should be descriptive and follow some rules:

Name a Memory Location

Rules for names (identifiers) in a C++ program

1. The name must begin with a letter.
2. The name can contain only letters, numbers, and the underscore. No punctuation characters, spaces, or special symbols (such as the dollar sign and percent sign) are allowed in the name.
3. The C++ compiler you are using determines the maximum number of characters in a name. Although names can contain thousands of characters, the recommended maximum number of characters to use is 32.
4. The name cannot be a keyword—such as the word `double`—because a keyword has a special meaning in C++. Appendix A contains a complete listing of the C++ keywords.
5. Names in C++ are case sensitive.

Valid names	Invalid names	
`deposit`	`98deposit`	(the name must begin with a letter)
`end_Balance`	`end Balance`	(the name cannot contain a space)
`withdrawal`	`withdrawal.amt`	(the name cannot contain punctuation)
`interest`	`int`	(the name cannot be a keyword)
`RATE`	`RATE%`	(the name cannot contain a percent sign)

Figure 4-3: How to name a memory location

Selecting a Name for a Memory Location (continued)

- Most programmers:
  - Use uppercase letters for named constants
  - Use lowercase letters for variables
  - Use camel case if a variable's name contains two or more words

Selecting a Name for a Memory Location (continued)

Memory location type	Name
variable	`radius`
variable	`radiusSquared`
variable	`area`
named constant	`PI`

Figure 4-4: Names of memory locations for the circle area problem

Selecting a Data Type for a Memory Location

Data type	Stores	Memory required	Values
`char`	one character	1 byte	one character
`short`	integer	2 bytes	–32,768 to 32,767
`int`	integer	4 bytes	–2,147,483,648 to 2,147,483,647
`float`	single precision floating-point number	4 bytes	–3.4e38 to 3.4e38 (7 digits of precision)
`double`	double precision floating-point number	8 bytes	–1.7e308 to 1.7e308 (15 digits of precision)
`string`	zero or more characters	1 byte per character	zero or more characters
`bool`	Boolean value	1 byte	`true, false`

Figure 4-5: Some of the data types available in C++

These data types, except `string`, are **fundamental data types**

Selecting a Data Type for a Memory Location (continued)

- `string` is a class
  - Program must include:
    - `#include <string>`
    - `using std::string;`
- C++ contains one or more data types for storing
  - Integers (whole numbers)
  - **Floating-point numbers** (with a decimal place)
  - **Characters** (letters, symbols, and numbers that will not be used in calculations)
  - **Boolean values** (true and false)

Selecting a Data Type for a Memory Location (continued)

- The data type to use for a memory location depends on the values it will store

Memory location	Name	Data type
variable	`radius`	`double`
variable	`radiusSquared`	`double`
variable	`area`	`double`
named constant	`PI`	`double`

Figure 4-6: Data type assigned to the memory locations for the circle area problem

How Data is Stored in Internal Memory

Decimal number system (base 10)		**Decimal number**	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
		110						1	1	0
		342						3	4	2
		31509				3	1	5	0	9
Binary number system (base 2)	**Binary number**	**Decimal equivalent of binary number**	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	110	6						1	1	0
	11010	26				1	1	0	1	0
	1001						1	0	0	1

Figure 4-7: Comparison of the decimal and binary number systems

How Data is Stored in Internal Memory (continued)

Character	ASCII	Binary	Character	ASCII	Binary	Character	ASCII	Binary
0	48	00110000	K	75	01001011	g	103	01100111
1	49	00110001	L	76	01001100	h	104	01101000
2	50	00110010	M	77	01001101	i	105	01101001
3	51	00110011	N	78	01001110	j	106	01101010
4	52	00110100	O	79	01001111	k	107	01101011
5	53	00110101	P	80	01010000	l	108	01101100
6	54	00110110	Q	81	01010001	m	109	01101101
7	55	00110111	R	82	01010010	n	110	01101110
8	56	00111000	S	83	01010011	o	111	01101111
9	57	00111001	T	84	01010100	p	112	01110000
:	58	00111010	U	85	01010101	q	113	01110001
;	59	00111011	V	86	01010110	r	114	01110010
A	65	01000001	W	87	01010111	s	115	01110011
B	66	01000010	X	88	01011000	t	116	01110100
C	67	01000011	Y	89	01011001	u	117	01110101
D	68	01000100	Z	90	01011010	v	118	01110110
E	69	01000101	a	97	01100001	w	119	01110111
F	70	01000110	b	98	01100010	x	120	01111000
G	71	01000111	c	99	01100011	y	121	01111001
H	72	01001000	d	100	01100100	z	122	01111010
I	73	01001001	e	101	01100101			
J	74	01001010	f	102	01100110			

Figure 4-8: Partial ASCII chart

Selecting an Initial Value for a Memory Location

- To **initialize** is to assign an initial value to a memory location
  - Typically a **literal constant**
    - Type can be: numeric, character, or string
  - A location with `bool` data type can be initialized with keywords `true` or `false`
  - Typical initialization values
    - 0
    - 0.0
    - ‘ ‘
    - “”
    - `true`, `false`

Selecting an Initial Value for a Memory Location (continued)

Numeric literal constants	Character literal constants	String literal constants
2 3.14 3.2e6 –2300 0	'X' '$' 'b' '2' ' ' (a space enclosed in single quotation marks)	"Hello" "Enter room length: " "450" "345AB" " " (two double quotation marks with no space between)

Important note: Notice that character literal constants are enclosed in single quotation marks, and string literal constants are enclosed in double quotation marks. Numeric literal constants, however, are not enclosed in any quotation marks.

Figure 4-9: Examples of numeric, character, and string literal constants

Type Conversions

- **Implicit type conversions** can occur when assigning a value to a memory location
  - Or, when processing calculation statements
  - Value can be **promoted** or **demoted**
    - Implicit demotion can adversely affect output
- Use **explicit type conversion** (**type casting**) to convert an item from one data type to another
  - **static_cast operator**

Type Conversions (continued)

Examples and results of implicit type conversions

Example 1

```
double price = 5.0;
double total = 0.0;
int quantity = 8;
total = price * quantity;
```

Result: assigns 40.0 to the `total` variable, which is correct

Example 2

```
int average  = 0;
double test1 = 90.0;
double test2 = 81.0;
average = (test1 + test2) / 2;
```

Result: assigns 85 to the `average` variable, which may or may not be what the programmer intended

Example 3

```
double average = 0.0;
int test1      = 90;
int test2      = 81;
average = (test1 + test2) / 2;
```

Result: assigns 85.0 to the `average` variable, which is not correct

Example 4

```
double average = 0.0;
int test1      = 90;
int test2      = 81;
average = (test1 + test2) / 2.0;
```

Result: assigns 85.5 to the `average` variable, which is correct

Figure 4-10: Calculation statements that trigger implicit type conversions

Type Conversions (continued)

Use the `static_cast` Operator

Syntax

```
static_cast<dataType>(data)
```

Example 1

```
float price = static_cast<float>3.1;
```

Result: converts the `double` number 3.1 to `float`, and then assigns the result to the `float` variable named `price`

Example 2

```
double average = 0.0;
int test1      = 90;
int test2      = 81;
average = static_cast<double>(test1 + test2) / 2.0;
```

Result: converts the sum of the two `int` variables to `double`, then divides the result by the `double` number 2.0, and then assigns the correct result (85.5) to the `double` variable named `average`

Example 3

```
double price = 5.0;
double total = 0.0;
int quantity = 8;
total = price *  static_cast<double>(quantity);
```

Result: converts to `double` the contents of the `int` variable named `quantity`, then multiplies the result by the contents of the `double` variable named `price`, and then assigns the correct result (40.0) to the `double` variable named `total`

Figure 4-11: How to use the `static_cast` operator

Variables and Named Constants (continued)

Memory location	Name	Data type	Initial value
variable	`radius`	`double`	0.0
variable	`radiusSquared`	`double`	0.0
variable	`area`	`double`	0.0
named constant	`PI`	`double`	3.141593

Figure 4-12: Initial values assigned to the memory locations for the circle area problem

Declaring a Named Constant

Declare a Named Constant

Syntax

const *dataType constantName* **=** *value*;

Examples

```
const double PI = 3.141593;
const float MAXPAY = static_cast<float>(15.75);
const int AGE = 65;
const bool PAID = true;
const char YES = 'Y';
const string TITLE = "IMG";
```

Figure 4-13: How to declare a named constant in C++

Declaring a Variable

Declare a Variable

Syntax

dataType variableName [= *initialValue*];

Examples

```
int age = 0;
float rate = 0.0;
float rate = static_cast<float>(3.5);
double sale = 0.0;
bool insured = false;
char grade = ' ';
string company = "";
```

Figure 4-14: How to declare a variable in C++

Declaring a Variable (continued)

```
double radius = 0.0;
double radiusSquared = 0.0;
double area = 0.0;
```

Figure 4-15: C++ statements reserving the `radius`, `radiusSquared`, and `area` variables

Using an Assignment Statement to Store Data in a Variable

Write an Assignment Statement

Syntax

variableName = *expression*;

In the examples, `hoursWkd`, `overtime`, and `age` are `int` variables, `cost` is a `float` variable, and `bonus` and `sales` are `double` variables. `RATE` is a `double` named constant, `name` is a `string` variable, `middleInitial` is a `char` variable, and `paid` is a `bool` variable.

Examples:

```
hoursWkd = 50;
overtime = hoursWkd - 40;
age = age + 1;
cost = static_cast<float>(6300.75);
bonus = sales * RATE;
name = "Jackie";
middleInitial = 'P';
paid = true;
```

Figure 4-16: How to write an assignment statement in C++

Using an Assignment Statement to Store Data in a Variable (continued)

C++ statements

```
int temp = 3;
temp = temp + 1;
temp = temp * 2;
```

How the statements are processed

- The declaration statement `int temp = 3;` creates the `temp` variable in the computer's internal memory and initializes the variable to the integer 3.
- The assignment statement `temp = temp + 1;` adds the number 1 to the contents of the `temp` variable, giving 4. It then replaces the 3 currently stored in the `temp` variable with the number 4. Notice that the computer evaluates the *expression* appearing on the right side of the assignment operator before assigning the result to the variable whose name appears on the left side. (All programming languages process assignment statements in the same manner.)
- The assignment statement `temp = temp * 2;` first multiplies the contents of the `temp` variable (4) by 2, giving 8. It then removes the number 4 from the `temp` variable and stores the number 8 there instead.

Figure 4-17: Assignment statements entered in a C++ program

Arithmetic Operators

- Precedence numbers indicate order in which computer performs the operation in an expression
  - Use parentheses to override order of precedence

Arithmetic Operators (continued)

Operator	Operation	Precedence number
( )	override normal precedence rules	1
–	negation	2
*, /, %	multiplication, division, and modulus arithmetic	3
+, –	addition and subtraction	4

Figure 4-18: Standard arithmetic operators and their order of precedence

Arithmetic Operators (continued)

Include Arithmetic Operators in Assignment Statements

In the following statements, `due` and `purchase` are `double` variables, and `quantity` and `total` are `int` variables. The `due` variable contains the number 0.0, the `purchase` variable contains the number 100.0, the `quantity` variable contains the number 10, and the `total` variable contains the number 0.

C++ statement	Result
`due = purchase - (purchase * .05);`	95.0
`due = purchase - purchase * .05;`	95.0
`total = quantity + 20 / 2;`	20
`total = (quantity + 20) / 2;`	15
`total = quantity % 4 * 3;`	6

Figure 4-19: How to include arithmetic operators in assignment statements

Arithmetic Assignment Operators

Use an Arithmetic Assignment Operator

Syntax

variableName arithmeticAssignmentOperator expression;

Operator	Purpose
+=	addition assignment
–=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	remainder assignment

Example1

Original statement: `rate = rate + .05;`

Abbreviated statement: `rate += .05;`

Example 2

Original statement: `price = price - discount;`

Abbreviated statement: `price -= discount;`

Figure 4-20: How to use an arithmetic assignment operator

Getting Data from the Keyboard

- Use >> to get numeric, character, or string values from the keyboard and store them in a variable
  - Stops reading characters when it encounters a **white-space character** in the input
    - Blank, tab, or newline
  - An alternative is to use **`getline()`**

The `getline()` Function

- When **`getline()`** encounters the delimiter character in the input, it **consumes** the character

Use the `getline()` Function to Get String Input from the Keyboard

Syntax

getline(cin, *stringVariableName*[, *delimiterCharacter*]);

Examples:

```
getline(cin, name);
getline(cin, name, '\n');
getline(cin, city, '#');
```

newline character

Figure 4-21: How to use the `getline()` function to get string input from the keyboard

Items between parentheses in a function's syntax are the **arguments**

The `ignore()` Function

- **`ignore()`** instructs computer to read and consume characters entered at keyboard

Use the `ignore()` Function

Syntax

cin.ignore([*numberOfCharacters*] [, *delimiterCharacter*]);

Examples:

```
cin.ignore(3);
cin.ignore(1);
cin.ignore();
cin.ignore(100, '\n');
cin.ignore(25, '#');
```

Figure 4-22: How to use the `ignore()` function

The ignore() Function (continued)

```
//Figure 4-23.cpp

#include <iostream>
#include <string>

using std::cout;
using std::cin;
using std::endl;
using std::string;

int main()
{
     //declare variables
     double sales     = 0.0;
     string stateName = "";

     //enter input items
     cout << "Enter the sales amount: ";
     cin >> sales;
     cin.ignore(100, '\n');
     cout << "Enter the state name: ";
     getline(cin, stateName);

     //display output items
     cout << "You entered sales of " << sales;
     cout << " for " << stateName << endl;

     return 0;
}    //end of main function
```

Figure 4-23: C++ program that requires the `ignore()` function *(Continued)*

Formatting Floating-Point Numbers

- Use **fixed stream manipulator** to display a floating-point number in fixed-point notation

```
#include <iostream>
using std::fixed;
```

- Use **scientific stream manipulator** for e notation

```
#include <iostream>
using std::scientific;
```

- **Setprecision stream manipulator** controls number of decimal places displayed

```
#include <iomanip>
using std::setprecision;
```

Formatting Floating-Point Numbers (continued)

Use the `fixed` and `scientific` Stream Manipulators

Example 1

```
double sales = 10575.25;
cout << fixed;
cout << sales << endl;
```

Result: displays 10575.250000

Example 2

```
double rate = 5.12345623;
cout << fixed << rate << endl;
```

Result: displays 5.123456

Example 3

```
double rate = 5.123456932;
cout << fixed << rate << endl;
```

Result: displays 5.123457

Example 4

```
double sales = 10575.25;
cout << scientific << sales << endl;
```

Result: displays 1.057525e+004

Figure 4-24: How to use the `fixed` and `scientific` stream manipulators

Formatting Floating-Point Numbers (continued)

Use the `setprecision` Stream Manipulator

Syntax

setprecision(*numberOfDecimalPlaces*)

Example 1 — Result

```
double sales = 3500.6;
cout << fixed;
cout << setprecision(2);
cout << sales << endl;
```

displays 3500.60

Example 2 — Result

```
double rate = 10.0732;
cout << fixed << setprecision(3);
cout << rate << endl;
```

displays 10.073

Example 3 — Result

```
double sales = 3467.55;
cout << fixed;
cout << setprecision(0) << sales;
```

displays 3468

Figure 4-25: How to use the `setprecision` stream manipulator

Summary

- Programs have variables, constants (named, literal), and arithmetic operators (to perform calculations)

  `const` *`dataType constantName = value`*`;`

  `dataType variableName` `[=` *`initialValue`*`];`
- Use assignment statement to store data in a variable

  `variableName = expression``;`
  - When assigning a value to a memory location, the value should fit the memory location's data type
    - Use `static_cast` operator to convert an item of data from one data type to another

Summary (continued)

- Arithmetic operators have a precedence number
  - Use parentheses to override order of precedence
- Arithmetic assignment operators abbreviate an assignment statement

  varName arithmeticAssignmentOp expr;
- `getline()` gets a string of characters
- `ignore()` reads and consumes characters entered at the keyboard
- `fixed` and `scientific` stream manipulators format the display of floating-point numbers