Lecture Notes on Jan/19/2023

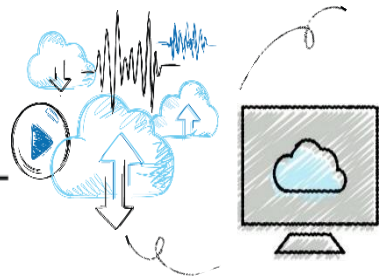
Chapters 1 and 2

An Overview of Computers and Programming Languages

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Processing a C++ Program (1 of 4)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;
    return 0;
}
```

Sample Run:

```
My first C++ program.
```

<iostream>: Standard Input / Output Streams Library Header that defines the standard input/output stream objects.

using namespace std: All the files in the C++ standard library declare all of its entities within the std namespace. That is why we have generally included the using namespace std; statement in all programs that used any entity defined in iostream.

Processing a C++ Program (2 of 4)

- Steps needed to process a C++ program
  1. Use a text editor to create the source code (source program) in C++
  2. Include preprocessor directives
     - Begin with the symbol # and are processed by the preprocessor
  3. Use the compiler to:
     - Check that the program obeys the language rules
     - Translate the program into machine language (**object program**)
  4. Use an integrated development environment (IDE) to develop programs in a high-level language
     - Programs such as mathematical functions are available
     - The library contains prewritten code you can use
     - A linker combines object program with other programs in the library to create executable code
  5. The loader loads executable program into main memory
  6. The last step is to execute the program

CENGAGE Learning

Processing a C++ Program (3 of 4)

- IDEs are quite user friendly
  - Compiler identifies the syntax errors and also suggests how to correct them
  - Build or Rebuild is a simple command that links the object code with the resources used from the IDE

Processing a C++ Program (4 of 4)

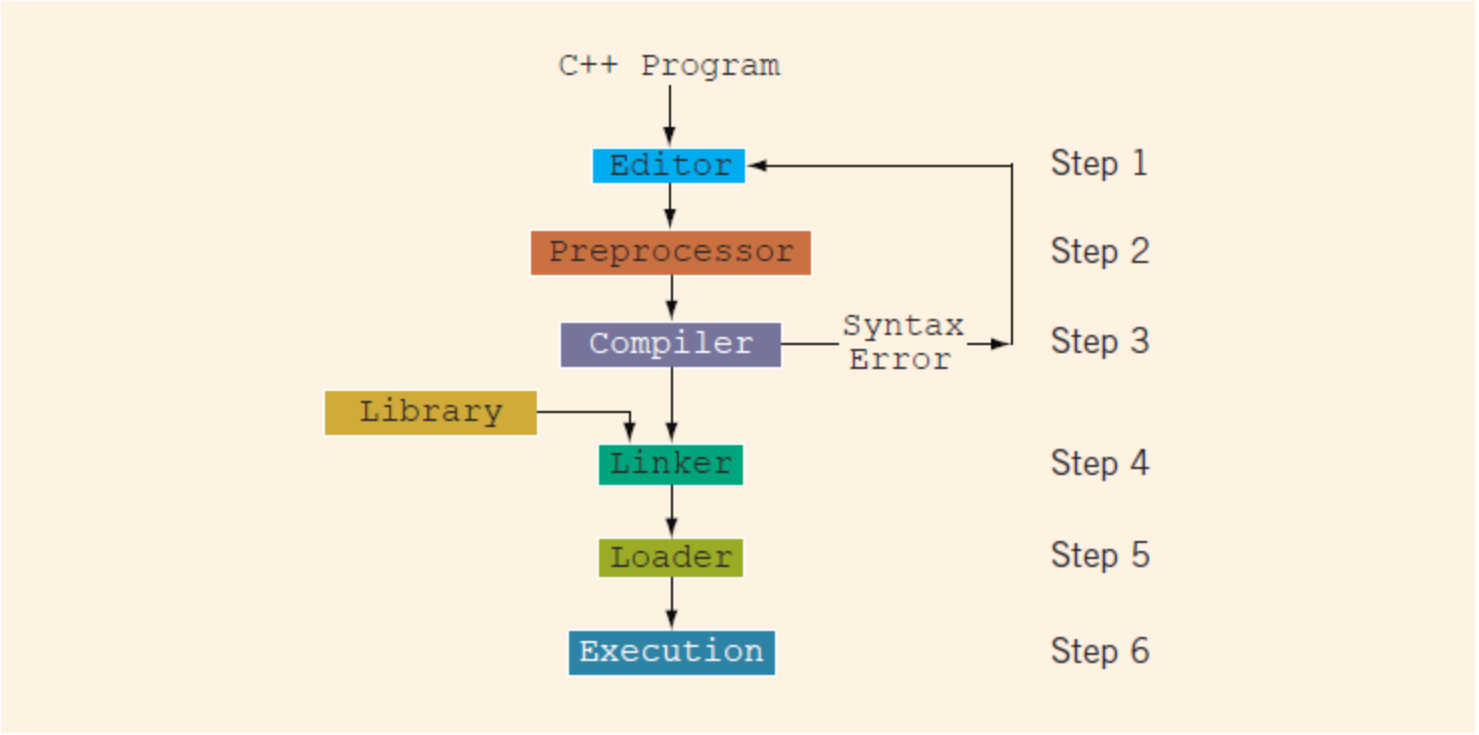


FIGURE 1-2 Processing a C++ program

Programming with the Problem Analysis–Coding–Execution Cycle

- Programming is a process of problem solving
- An algorithm is a step-by-step problem-solving process
  - A solution is achieved in a finite amount of time

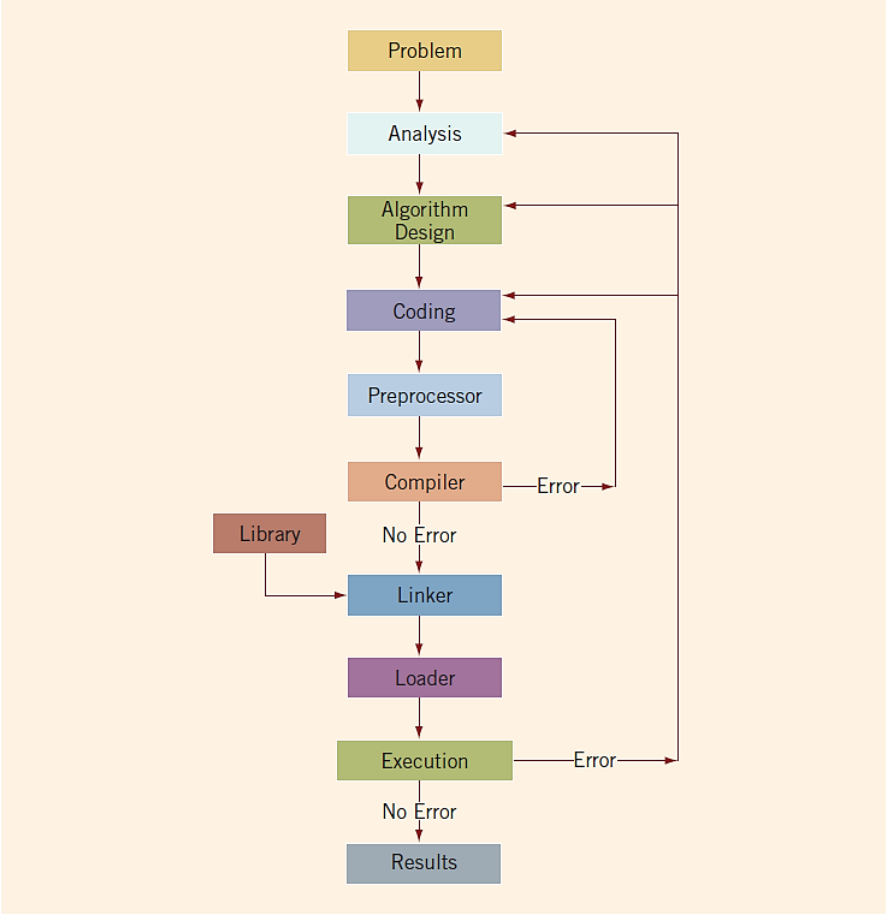


FIGURE 1-3 Problem analysis–coding–execution cycle

The Problem Analysis–Coding–Execution Cycle (1 of 5)

- Step 1: Analyze the problem
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem
- Step 2: Implement the algorithm
  - Implement the algorithm in code
  - Verify that the algorithm works
- Step 3: Maintain the program
  - Use and modify the program if the problem domain changes

The Problem Analysis–Coding–Execution Cycle (2 of 5)

- Analyze the problem using these steps:
  - Step 1: Thoroughly understand the problem and all requirements
  - Step 2: Understand the problem requirements
    - Does program require user interaction?
    - Does program manipulate data?
    - What is the output?
  - Step 3: If complex, divide the problem into subproblems
    - Analyze and design algorithms for each subproblem
- Check the correctness of algorithm
  - Test the algorithm using sample data
  - Some mathematical analysis might be required

The Problem Analysis–Coding–Execution Cycle (3 of 5)

- Once the algorithm is designed and correctness is verified
  - Write the equivalent code in high-level language
- Enter the program using a text editor

The Problem Analysis–Coding–Execution Cycle (4 of 5)

- Run code through the compiler
- If compiler generates errors
  - Look at code and remove errors
  - Run code again through compiler
- If there are no syntax errors
  - Compiler generates equivalent machine code
- Link machine code with the system's resources
  - Performed by the linker

The Problem Analysis–Coding–Execution Cycle (5 of 5)

- Once compiled and linked, the loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language
  - Does not guarantee that the program will run correctly

CENGAGE Learning

Algorithm Design: Example 1-1 from Book (1 of 2)

- Design an algorithm to find the perimeter and area of a rectangle
- The perimeter and area of the rectangle are given by the following formulas:

```
perimeter = 2 * (length + width)
area = length * width
```

Algorithm Design: Example 1-1 from Book (2 of 2)

- Algorithm
  - Get the length of the rectangle
  - Get the width of the rectangle
  - Find the perimeter with this equation:

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

  - Find the area with this equation:

    ```
    area = length * width
    ```

CENGAGE Learning

Programming Methodologies

- Two popular approaches to programming design
  - Structured
  - Object-oriented

Structured Programming

- Structured design
  - Involves dividing a problem into smaller subproblems
- Structured programming
  - Involves implementing a structured design
- The structured design approach is also called:
  - Top-down (or bottom-up) design
  - Stepwise refinement
  - Modular programming

CENGAGE Learning

Object-Oriented Programming (1 of 3)

- Object-oriented design (OOD)
  - Identify components called objects
  - Determine how objects interact with each other
- Specify relevant data and possible operations to be performed on that data
- Each object consists of data and operations on that data

Object-Oriented Programming (2 of 3)

- An object combines data and operations on the data into a single unit
- A programming language that implements OOD is called an object-oriented programming (OOP) language
- To design and use objects, you must learn how to:
  - Represent data in computer memory
  - Manipulate data
  - Implement operations

Object-Oriented Programming (3 of 3)

- To create operations:
  - Write algorithms and implement them in a programming language
  - Use functions to implement algorithms
- Learn how to combine data and operations on the data into a single unit called a class
- C++ was designed to implement OOD
- OOD is used with structured design

ANSI/ISO Standard C++

- C++ evolved from C
- C++ designed by Bjarne Stroustrup at Bell Laboratories in early 1980s
  - Many different C++ compilers were available
- C++ programs were not always portable from one compiler to another
- In mid-1998, ANSI/ISO C++ language standards were approved
- Second standard, called C++11, was approved in 2011

A Quick Look at a C++ Program (1 of 5)

EXAMPLE 2-1

```
//*************************************************************
// 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;
    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;
}
```

Single-precision floating-point format (sometimes called FP32 or float32) is a computer number format, usually occupying 32 bits in computer memory; it represents a wide dynamic range of numeric values by using a floating radix point.

Double-precision floating-point format (sometimes called FP64 or float64) is a floating-point number format, usually occupying 64 bits in computer memory; it represents a wide dynamic range of numeric values by using a floating radix point.

A Quick Look at a C++ Program (2 of 5)

- Sample Run:

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

CENGAGE Learning

A Quick Look at a C++ Program (3 of 5)

```
//************************************************************
// 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**.

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

A Quick Look at a C++ Program (4 of 5)

```
    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 (cont'd.)

A Quick Look at a C++ Program (5 of 5)

- Variable: a memory location whose contents can be changed

FIGURE 2-3 Memory allocation

FIGURE 2-4 Memory spaces after the statement `length = 6.0;` executes

The Basics of a C++ Program

- Subprogram (or function): collection of statements
  - When executed, accomplishes something
  - May be predefined or standard
- 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

CENGAGE Learning

Comments

- Comments are for the reader, not the compiler
- Two types
  - Single line: begins with `//`

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

  - Multiple line: enclosed between `/*` and `*/`

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

CENGAGE Learning

Special Symbols

- A token is the smallest individual unit of a program written in any language
- C++ tokens include special symbols, word symbols, and identifiers
- Special symbols in C++ include:

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

Reserved Words (Keywords)

- Reserved word symbols (or keywords):
  - Cannot be redefined within a program
  - Cannot be used for anything other than their intended use
- Examples include:
  - `int`
  - `float`
  - `double`
  - `char`
  - `const`
  - `void`
  - `return`

Identifiers (1 of 2)

- An identifier is the name of something that appears in a program
  - Consists of letters, digits, and the underscore character (_)
  - Must begin with a letter or underscore
- C++ is case sensitive
  - **`NUMBER`** is not the same as **`number`**
- Two predefined identifiers are **`cout`** and **`cin`**
- Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea

CENGAGE Learning

Identifiers (2 of 2)

- Legal identifiers in C++
  - first
  - conversion
  - payRate

TABLE 2-1 Examples of Illegal Identifiers

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

Whitespaces

- Every C++ program contains whitespaces
  - Include blanks, tabs, and newline characters
- Whitespaces separate special symbols, reserved words, and identifiers
- Proper utilization of whitespaces is important
  - Can be used to make the program more readable

Data Types

- A data type is set of values together with a set of allowed operations
- C++ data types fall into three categories:
  - Simple data type
  - Structured data type
  - Pointers

CENGAGE Learning

Simple Data Types (1 of 2)

- Three categories of simple data
  - Integral: integers (numbers without a decimal)
    - Can be further categorized: `char`, `short`, `int`, `long`, `bool`, `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long`
  - Floating-point: decimal numbers
  - Enumeration: a user-defined data type

Simple Data Types (2 of 2)

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

Data Type	Values	Storage (in bytes)
`int`	–147483648 (= -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

`int` Data Type

- Examples
  - `-6728`
  - `0`
  - `78`
  - `+763`
- Positive integers do not require a **+** sign
- A comma cannot be used within an integer
  - Commas are only used for separating items in a list

CENGAGE Learning

`bool` Data Type

- **`bool`** type
  - Two values: **`true`** and **`false`**
  - Purpose: to manipulate logical (Boolean) expressions
- **`true`** and **`false`**
  - Logical values
- **`bool`**, **`true`**, and **`false`**
  - Reserved words

`char` Data Type (1 of 2)

- Data type **`char`** is the smallest integral data type
- It is used for single characters: letters, digits, and special symbols
- Each character is enclosed in single quotes
  - **`'A', 'a', '0', '*', '+', '$', '&'`**
- A blank space is a character
  - Written `' '`, with a space left between the single quotes

`char` Data Type (2 of 2)

- 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

Floating-Point Data Types (1 of 3)

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

TABLE 2-3 Examples of Decimal Numbers in Scientific and C11 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 (2 of 3)

- `float`: represents any real number
  - Range: `-3.4 * 10`^{38} to `3.4 * 10`^{38} (four bytes)
- `double`: represents any real number
  - Range: `-1.7 * 10`^{308} to `1.7 * 10`^{308} (eight bytes)
- Minimum and maximum values of data types are system dependent

Floating-Point Data Types (3 of 3)

- Maximum number of significant digits (decimal places) for `float` values: 6 or 7
- Maximum number of significant digits for `double`: 15
- Precision: maximum number of significant digits
  - `float` values are called single precision
  - `double` values are called double precision

Data Types, Variables, and Assignment Statements

- To declare a variable, must specify its data type
- Syntax rule to declare a variable is:
  - **`dataType identifier;`**
- Examples include:

```
int counter;
double interestRate;
char grade;
```

- Assignment statement has the form: **`variable = expression`**
  - Example: **`interestRate = 0.05;`**

Arithmetic Operators, Operator Precedence, and Expressions (1 of 2)

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

Arithmetic Operators, Operator Precedence, and Expressions (2 of 2)

- When you use / with integral data types, the integral result is truncated (no rounding)
- Arithmetic expressions contain values and arithmetic operators
- Operands are the numbers appearing in the expressions
- Operators can be unary (one operand) or binary (two operands)

CENGAGE Learning

Order of Precedence

- All operations inside `()` 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
- When operators are on the same level
  - Operations are performed from left to right (associativity)
- `3 * 7 - 6 + 2 * 5 / 4 + 6` means
  `(((3 * 7) - 6) + ((2 * 5) / 4 )) + 6`

Expressions

- Integral expression: all operands are integers
  - Yields an integral result
  - Example: `2 + 3 * 5`
- Floating-point (decimal) expression: all operands are floating-point
  - Yields a floating-point result
  - Example: `12.8 * 17.5 – 34.50`

Mixed Expressions (1 of 2)

- 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 (2 of 2)

- Evaluation rules
  - If operator has same types of operands
    - The operator is evaluated according to the type of the operands
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

Type Conversion (Casting) (1 of 2)

- Implicit type coercion: when the value of one type is automatically changed to another type
- Cast operator (also called type conversion or type casting): provides explicit type conversion
  - `static_cast<dataTypeName>(expression)`

Type Conversion (Casting) (2 of 2)

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`

`string` Type

- Data type `string` is a programmer-defined type supplied in ANSI/ISO Standard C++ library
- A string is a sequence of zero or more characters enclosed in double quotation marks
- A null (or empty) string is a string with no characters
- Each character has a relative position in the string
  - Position of first character is **0**
- The length of a string is the number of characters in it
  - Example: length of **`"William Jacob"`** is 13

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:
  1. Instruct the computer to allocate memory
  2. Include statements in the program to put data into the allocated memory

Allocating Memory with Constants and Variables (1 of2)

- Named constant: memory location whose content cannot 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 (2 of 2)

- Variable: memory location whose content may change during execution
- Syntax to declare one or multiple variables

```
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 (1 of 4)

- 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 (2 of 4)

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.";
```

Assignment Statement (3 of 4)

- Example 2-14 illustrates a walk-through (tracing values through a sequence)

	Values of the Variables/Statement	Explanation
Before Statement 1	num1: ? num2: ? num3: ?	
After Statement 1	num1: 18 num2: ? num3: ? `num1 = 18;`	
After Statement 2	num1: 45 num2: ? num3: ? `num1 = num1 + 27;`	**num1 + 27 = 18 + 27 = 45.** This value is assigned to **num1**, which replaces the old value of **num1**.
After Statement 3	num1: 45 num2: 45 num3: ? `num2 = num1;`	Copy the value of **num1** into **num2**.
After Statement 4	num1: 45 num2: 45 num3: 9 `num3 = num2 / 5;`	**num2 / 5 = 45 / 5 = 9.** This value is assigned to **num3**. So **num3 = 9.**
After Statement 5	num1: 45 num2: 45 num3: 2 `num3 = num3 / 4;`	**num3 / 4 = 9 / 4 = 2.** This value is assigned to num3, which replaces the old value of **num3**.

Assignment Statement (4 of 4)

- Given `int` variables `x`, `y`, and `z`. How is this legal C++ statement evaluated?

```
x = y = z
```

- The assignment operator is evaluated from right to left
  - The associativity of the assignment operator is from right to left

Saving and Using 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 and Initializing Variables

- Not all types of variables are initialized automatically
- 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

CENGAGE Learning

Input (Read) Statement (1 of 3)

- **`cin`** is used with **>>** to gather one or more inputs

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

- This is called an input (read) statement
- The stream extraction operator is **>>**
- For example, if miles is a **`double`** variable:
  `cin >> miles;`
  - Causes the computer to get a value of type double and places it in the variable **`miles`**

Input (Read) Statement (2 of 3)

- 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

Input (Read) Statement (3 of 3)

EXAMPLE 2-17

Suppose we have the following statements:

```
int feet;
int inches;
```

Suppose the input is:

```
23 7
```

Next, consider the following statement:

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

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++;
```

Output (1 of 4)

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

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

  - Called an output statement
- The stream insertion operator is **<<**
- Expression evaluated and its value is printed at the current cursor position on the screen

Output (2 of 4)

- A manipulator is used to format the output
  - Example: **`endl`** causes the 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.`

CENGAGE Learning

Output (3 of 4)

- The new line character (new line escape sequence) is **'\n'**
  - May appear anywhere in the string
- Examples

```
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 (4 of 4)

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

Preprocessor Directives (1 of 2)

- 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 processed by the preprocessor program
- All preprocessor commands begin with **#**
- No semicolon is placed at the end of these commands

Preprocessor Directives (2 of 2)

- 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