

# Chapter 1:

## Introduction to Computers and Programming



Addison-Wesley  
is an imprint of

PEARSON

Copyright © 2012 Pearson Education, Inc.

# 1.1

## Why Program?



# Why Program?

## Computer

programmable machine designed to follow instructions

## Program

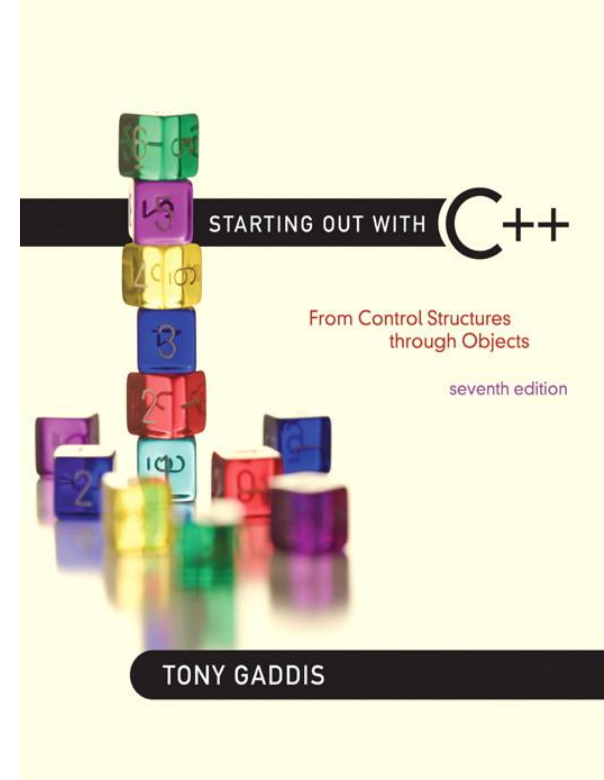
instructions in computer memory to make it do something

## Programmer

person who writes instructions (programs) to make computer perform a task

SO, without programmers, no programs; without programs, a computer cannot do anything

1.2

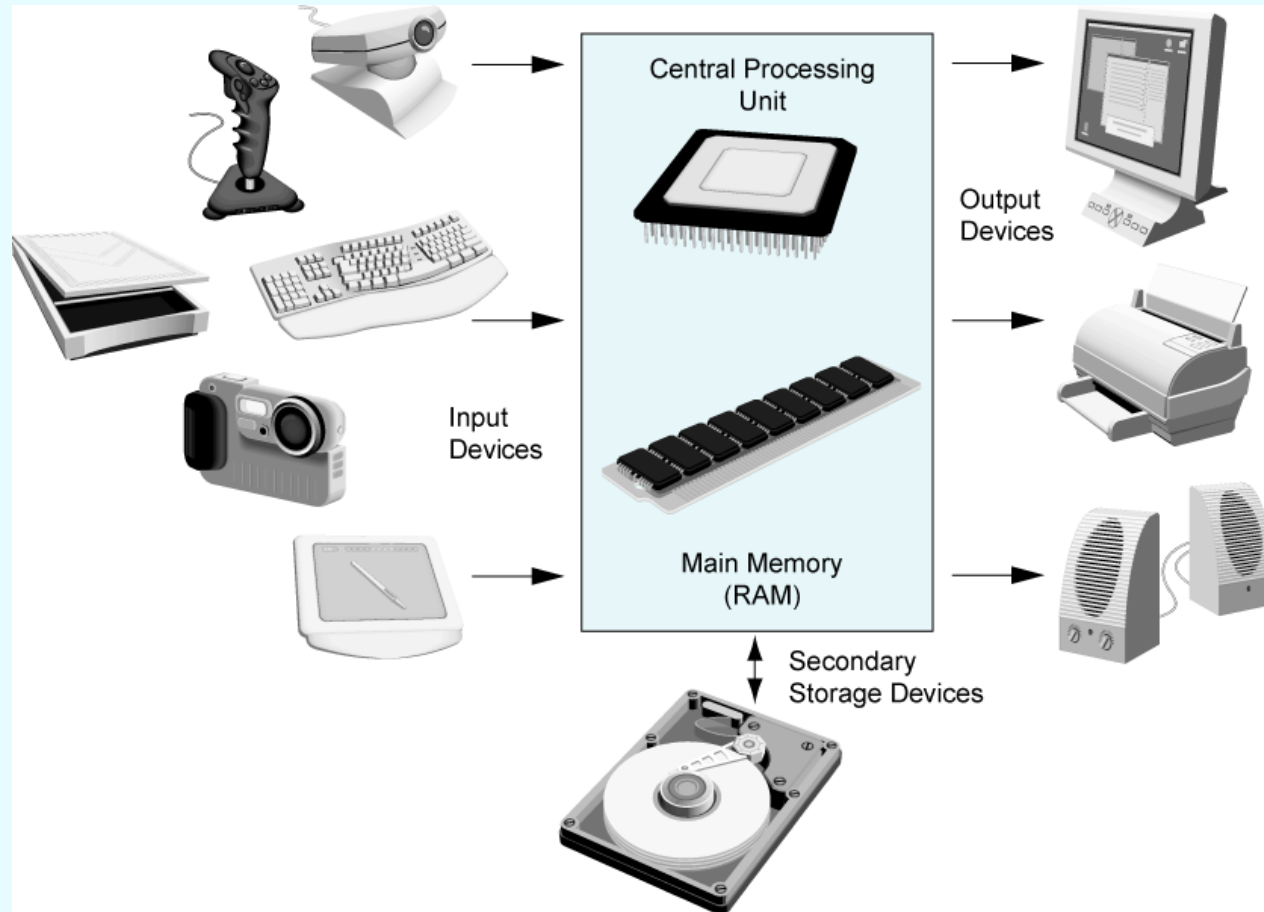


# Computer Systems: Hardware and Software

# Main Hardware Component Categories:

1. Central Processing Unit (CPU)
2. Main Memory ( aka RAM )
3. Secondary Memory / Storage
4. Input Devices
5. Output Devices

# Main Hardware Component Categories



**Figure 1-2**

# Central Processing Unit (CPU)

Comprised of:

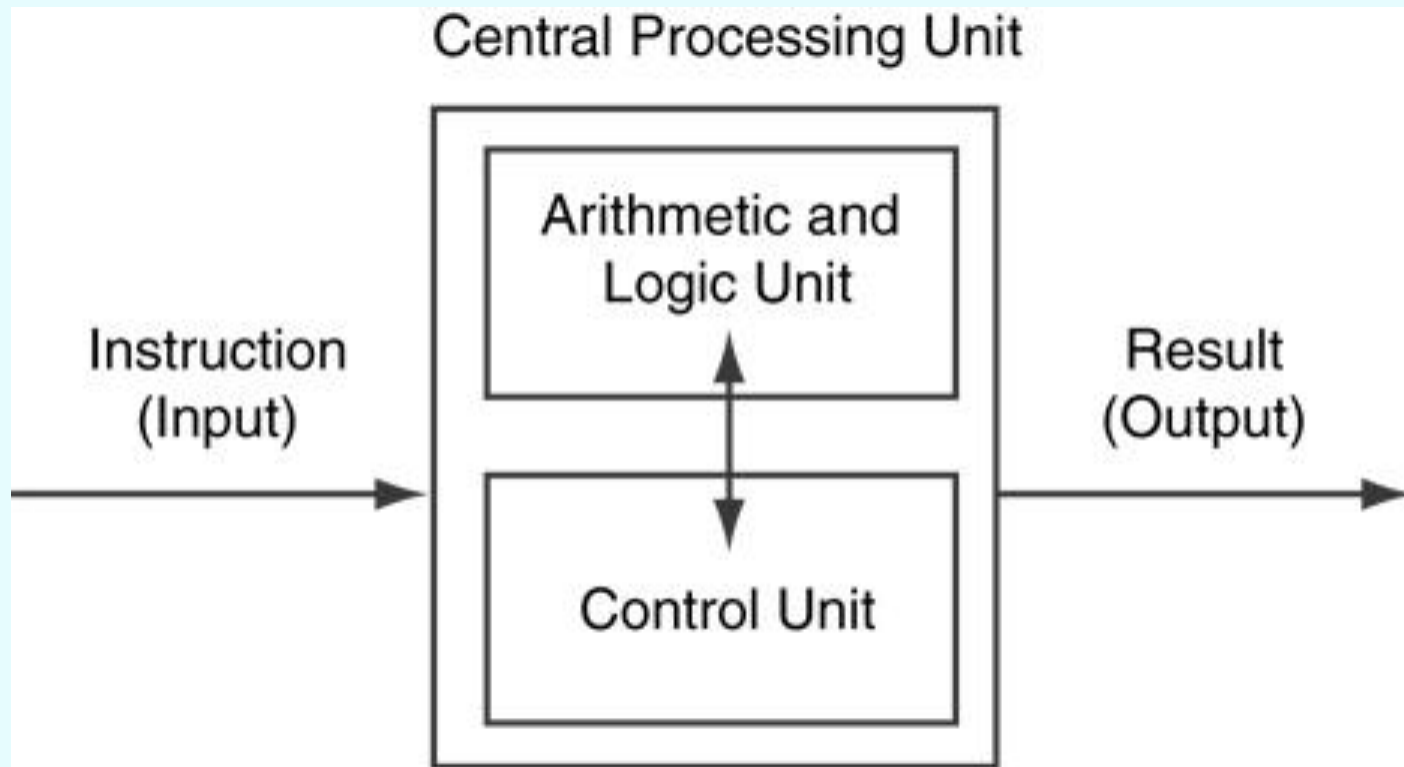
- Control Unit ( CU )

- Retrieves and decodes program instructions
- Coordinates activities of all other parts of computer

- Arithmetic & Logic Unit ( ALU )

- Hardware optimized for high-speed numeric calculation
- Hardware designed for true/false, yes/no decisions

# CPU Organization



**Figure 1-3**



# Main Memory

- It is **volatile**. Main memory is erased when a program terminates or computer is turned off
- Also called Random Access Memory (RAM)
- Organized as follows:
  - bit:
    - smallest piece of memory. **Has values** 0 (off, false) or 1 (on, true)
  - byte:
    - 8 consecutive bits. **Bytes have addresses.**

# Main Memory

- Address
  - Each byte in memory is identified by a unique number known as an **address**.

# Main Memory

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	149	17	18
19	20	21	22	23	72	24	25	26	27
28	29								

- In Figure 1-4, the number 149 is stored in the byte with the address 16, and the number 72 is stored at address 23.

# Secondary Storage

- Non-volatile: data retained when program is not running or computer is turned off
- Comes in a variety of media:
  - magnetic: floppy disk, hard drive
  - optical: CD-ROM, DVD
  - Flash drives, connected to the USB port

# Input Devices

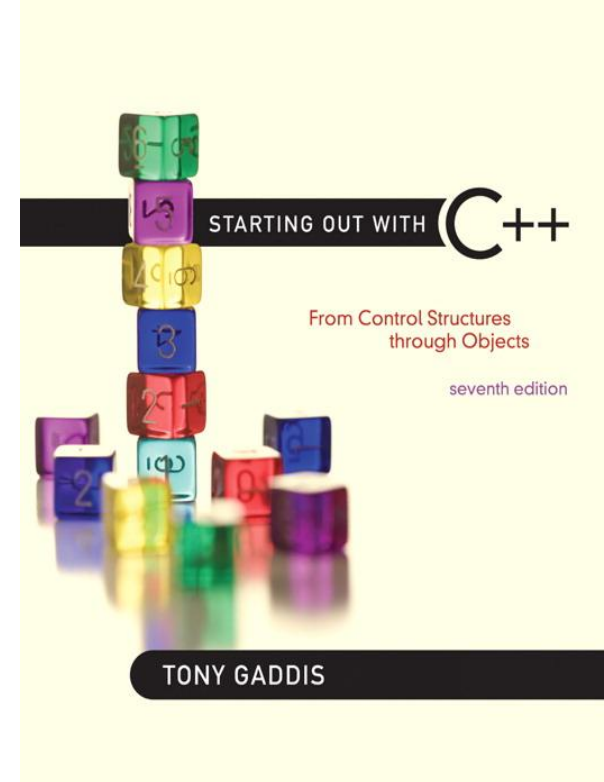
- Devices that send information to the computer from outside
- Many devices can provide input:
  - Keyboard, mouse, scanner, digital camera, microphone
  - Disk drives, CD drives, and DVD drives

# Software-Programs That Run on a Computer

- Categories of software:
  - System software:
    - programs that manage the computer hardware and the programs that run on them.
    - Examples:
      - operating systems, utility programs, software development tools
  - Application software:
    - programs that provide services to the user.
    - Examples:
      - word processing, games, programs to solve specific problems

# 1.3

## Programs and Programming Languages



# Programs and Programming Languages

- A program is a set of instructions that the computer follows to perform a task
- We start with an **algorithm**, which is a set of well-defined steps.



# Example Algorithm for Calculating Gross Pay

1. Display a message on the screen asking “How many hours did you work?”
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking “How much do you get paid per hour?”
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the amount paid per hour, and store the result in memory.
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in Step 5.

# Machine Language

- Although the previous algorithm defines the steps for calculating the gross pay, it is not ready to be executed on the computer.
- The computer only executes machine language instructions

# Machine Language

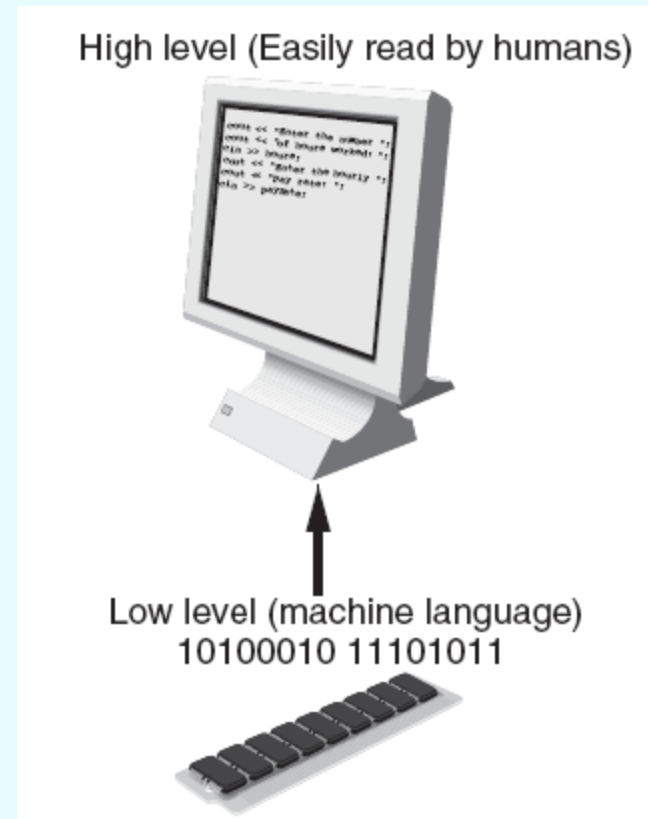
- Machine language instructions are **binary numbers**, such as

1011010000000101

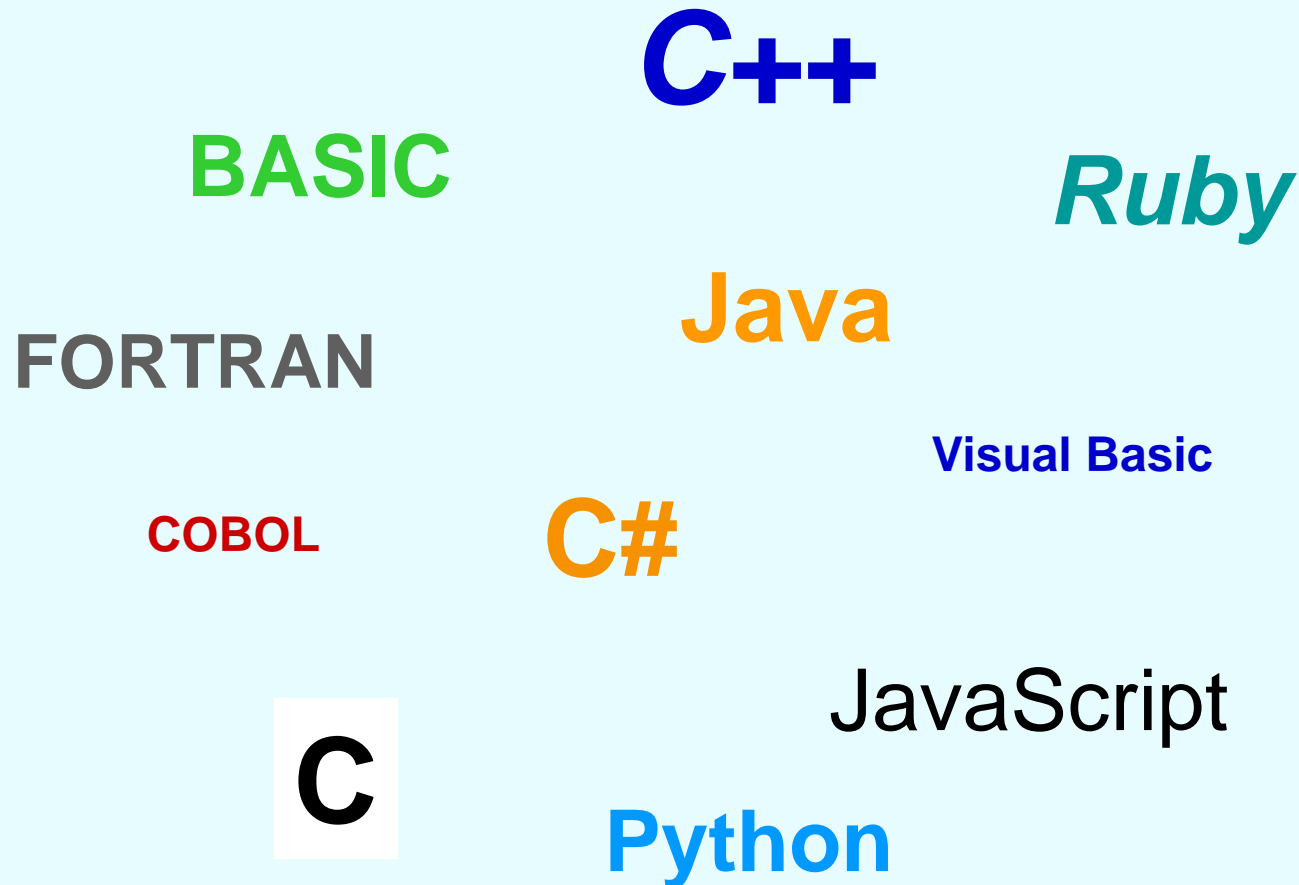
- Rather than writing programs in machine language, programmers use **programming languages**.

# Programs and Programming Languages

- Types of languages:
  - Low-level:
    - used for communication with computer hardware directly. Often written in binary machine code (0's/1's) directly.
  - High-level:
    - closer to human language



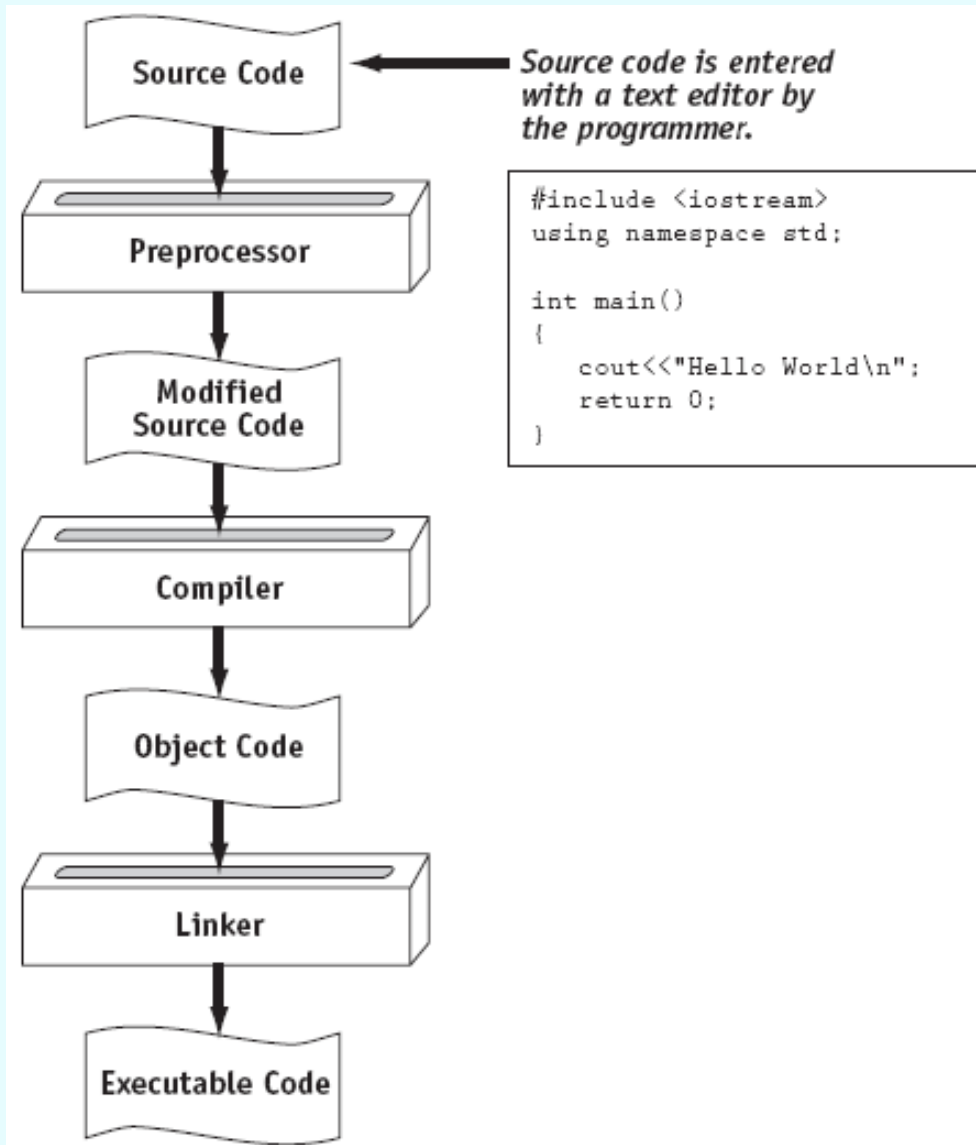
# Some Well-Known Programming Languages (Table 1-1 on Page 10)



# From a High-Level Program to an Executable File

- a) Create file containing the program with a text editor.
  - b) Run preprocessor to convert source file directives to source code program statements.
  - c) Run compiler to convert source program into machine instructions.
  - d) Run linker to connect hardware-specific code to machine instructions, producing an executable file.
- ✓ Steps b–d are often performed by a single command or button click.
  - ✓ Errors detected at any step will prevent execution of following steps.

# From a High-Level Program to an Executable File

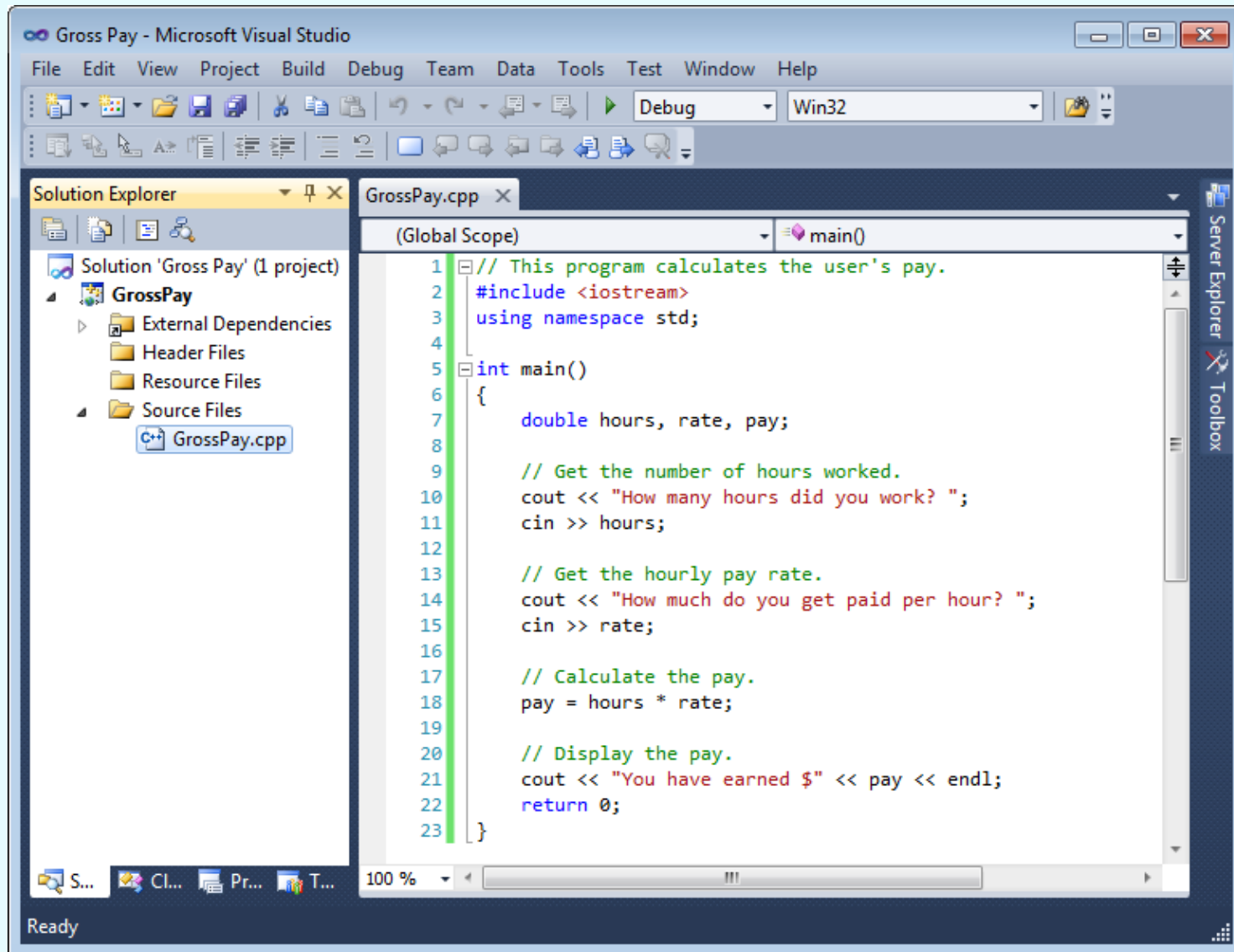


# Integrated Development Environments (IDEs)

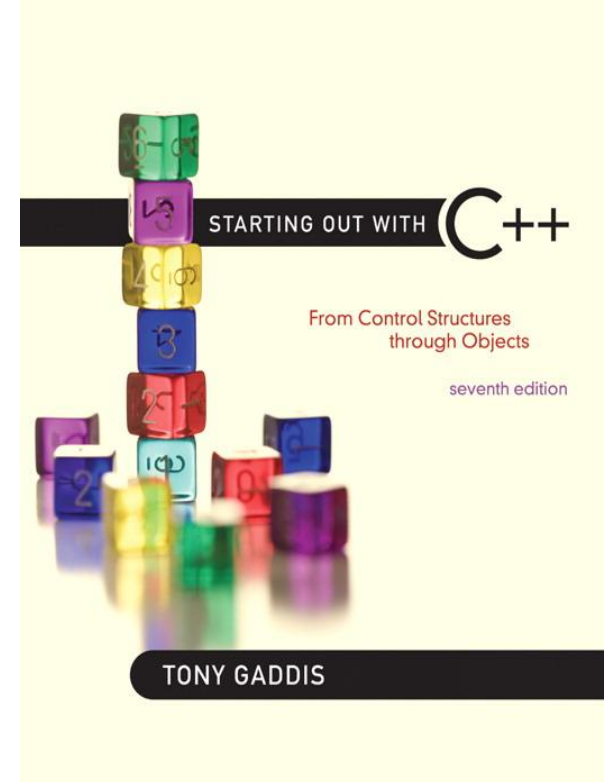
- An integrated development environment, or IDE, combines all the tools needed to **write**, **compile**, and **debug** a program into a single software application.
- Examples are Microsoft Visual C++, Turbo C++ Explorer, Eclipse, NetBeans, jGRASP, CodeWarrior, etc.



# Integrated Development Environments (IDEs)



# 1.4



## What is a Program Made of?

# What is a Program Made of?

- Common elements in programming languages:
  - Key Words
  - Programmer-Defined Identifiers
  - Operators
  - Punctuation
  - Syntax

# Program 1-1

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      double hours, rate, pay;
8
9      // Get the number of hours worked.
10     cout << "How many hours did you work? ";
11     cin >> hours;
12
13     // Get the hourly pay rate.
14     cout << "How much do you get paid per hour? ";
15     cin >> rate;
16
17     // Calculate the pay.
18     pay = hours * rate;
19
20     // Display the pay.
21     cout << "You have earned $" << pay << endl;
22     return 0;
23 }
```

# Key Words

- Also known as reserved words
- Have a special meaning in C++
- Can not be used for any other purpose
- Key words in the Program 1-1: `using`, `namespace`, `int`, `double`, and `return`

# Key Words

```
1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

# Programmer-Defined Identifiers

- Names made up by the programmer
- Not part of the C++ language
- Used to represent various things: variables (memory locations), functions, etc.
- In Program 1-1: `hours`, `rate`, and `pay`.

# Operators

- Used to perform operations on data
- Many types of operators:
  - Arithmetic:  $+$   $-$   $*$   $/$   $\%$
  - Assignment:  $=$
- Some operators in Program1-1:  
 $<<$   $>>$   $=$   $*$



# Operators

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      double hours, rate, pay;
8
9      // Get the number of hours worked.
10     cout << "How many hours did you work? ";
11     cin >> hours;
12
13     // Get the hourly pay rate.
14     cout << "How much do you get paid per hour? ";
15     cin >> rate;
16
17     // Calculate the pay.
18     pay = hours * rate;
19
20     // Display the pay.
21     cout << "You have earned $" << pay << endl;
22     return 0;
23 }
```

# Punctuation

- Characters that mark the end of a statement, or that separate items in a list
- In Program 1-1: , and ;

# Punctuation

```
1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

# Syntax

- The rules of grammar that must be followed when writing a program
- Controls the use of key words, operators, programmer-defined symbols, and punctuation

# Variables

- A variable is a named storage location in the computer's memory for holding a piece of data.
- In Program 1-1 we used three variables:
  - The `hours` variable was used to hold the hours worked
  - The `rate` variable was used to hold the pay rate
  - The `pay` variable was used to hold the gross pay

# Variable Definitions

- To create a variable in a program you must write a variable definition (also called a variable declaration)
- Here is the statement from Program 1-1 that defines the variables:

```
double hours, rate, pay;
```

# Variable Definitions

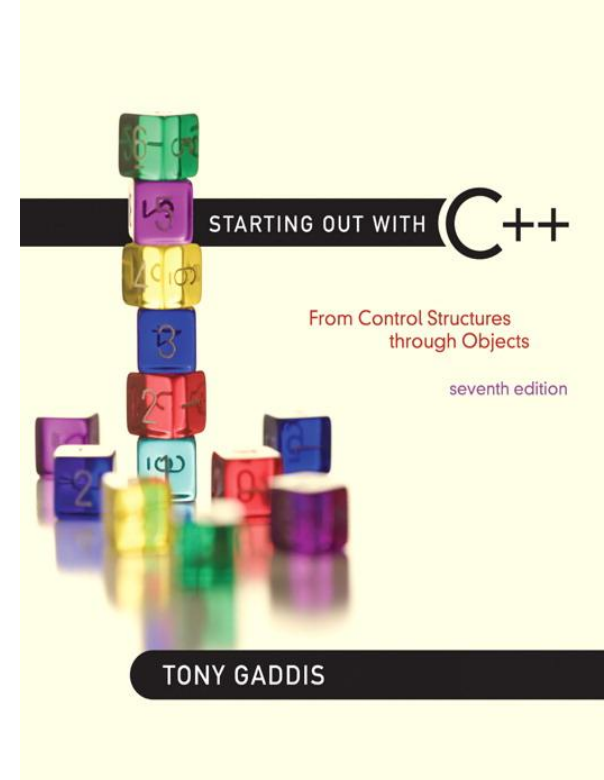
- There are many different types of data, which you will learn about in this course.
- A variable holds a specific type of data.
- The variable definition specifies the type of data a variable can hold, and the variable name.

# Variable Definitions

- Once again, line 7 from Program 1-1:  
**double hours, rate, pay;**
- The word **double** specifies that the variables can hold double-precision floating-point numbers. (You will learn more about that in Chapter 2)



# 1.5



## Input, Processing, and Output

# Input, Processing, and Output

Three steps that a program typically performs:

**1) Gather **input** data:**

- from keyboard
- from files on disk drives

**2) **Process** the input data**

**3) Display the results as **output**:**

- send it to the screen
- write to a file

1.6



## The Programming Process

# The Programming Process

1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.
5. Type the code, save it, and compile it.
6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

1.7



# Procedural and Object-Oriented Programming

# Procedural and Object-Oriented Programming

- **Procedural** programming:
  - focus is on the process.
  - Procedures/functions are written to process data.
- **Object-Oriented** programming:
  - focus is on objects, which contain data and the means to manipulate the data.
  - Messages are sent to objects to perform operations.