UEE1302 Introduction to Computers and Programming

C_Lecture 01:

C Basic

C: How to Program 7th ed.

Agenda

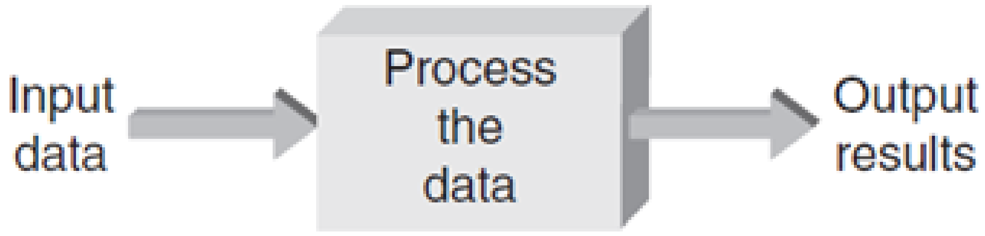
- Introduction to Programming
- Algorithms and Procedures
- Typical C Development Environment
- First Program in C
- Memory Concepts
- Arithmetic, Equality and Relational Operators
- Style of C Programs

Introduction to Programming

- Computer program
  - Data and instructions used to operate a computer
- Programming
  - Writing computer program in a language that the computer can respond to and that other programmers can understand
- Programming language
  - Set of instructions, data, and rules used to construct a program
    - Machine languages and assembly languages
    - High-level languages

Introduction to Programming (cont.)

- Procedural language
  - Instructions are used to create self-contained units (procedures)
  - Procedures accept data as input and transform data to produce a specific result as an output
  - Initially, high-level programming languages were predominately procedural



Introduction to Programming (cont.)

- Object-oriented languages
  - Program must define objects that it is manipulating
  - Such definitions include:
    - The general characteristics of objects
    - Specific operations to manipulate objects
- C++ is an object-oriented language
  - Has procedures and objects
  - Supports code reuse

Introduction to Programming (cont.)

- C++ began as extension to C
  - C is a procedural language developed in the 1970s at AT&T Bell Laboratories
- In early 1980s, Bjarne Stroustrup (also at AT&T) used his background in simulation languages to develop C++
- Object-orientation and other procedural improvements were combined with existing C language features to form C++

Algorithms and Procedures

- Before writing a program, a programmer must clearly understand:
  - What data is to be used
  - The desired result
  - The procedure needed to produce this result
- The procedure is referred to as an algorithm
- Algorithm: Step-by-step sequence of instructions describing how to perform a computation

Example of Algorithm

- Assume that a program must calculate the sum of all whole numbers from 1 through 100
- A computer:
  - Cannot respond to heuristic command: "Add the numbers from 1 - 100"
  - Is an *algorithm*-responding machine and not a heuristic-responding machine
- Several methods or algorithms can be used to find the required sum

Example of Algorithm (cont.)

- Method 1: Columns – Arrange the numbers from 1 to 100 and add them

- Method 2: Formula – use the formula

$$sum = \frac{n(a+b)}{2}$$

n: number of terms to be added

a: first number to be added

b: last number to be added

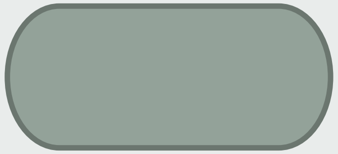
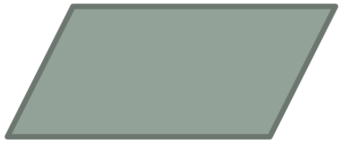
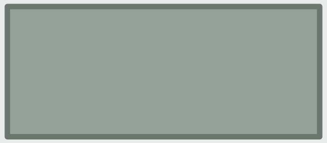
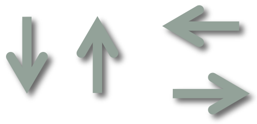
Pseudocode

- Pseudocode: the English-like phrases that are used to describe the steps in an algorithm

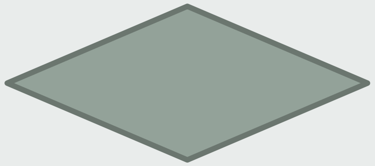
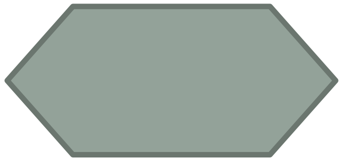
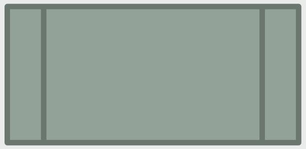
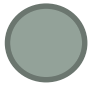
$$sum = \frac{n(a+b)}{2}$$

```
Step 1:
    set n equal to 100
Step 2:
    set a equal to 1
Step 3:
    set b equal to 100
Step 4:
    calculate sum = n(a+b)/2
Step 5:
    print the sum
```

Flowchart Symbols

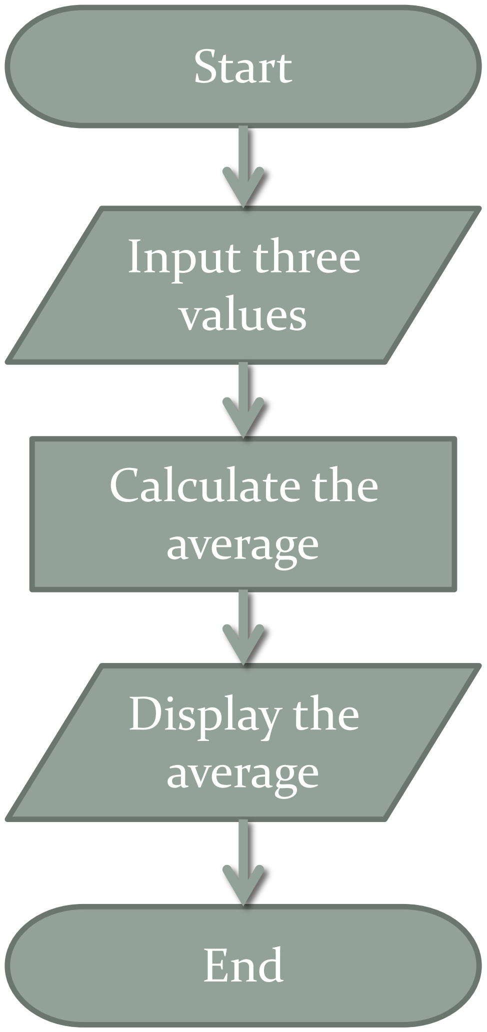
Symbol	Name	Description
	Terminal	Indicates the beginning or end of an algorithm
	Input/Output	Indicates an input or output operation
	Process	Indicates computation or data manipulation
	Flow lines	Used to connect the flowchart symbols and indicate the logic flow

Flowchart Symbols (cont.)

Symbol	Name	Description
	Decision	Indicates a decision point in the algorithm
	Loop	Indicates the initial, final and increment values of a loop
	Predefined Process	Indicates a predefined process, as in calling a sorting process
	Connector	Indicates an entry to, or exit from another part of the flowchart

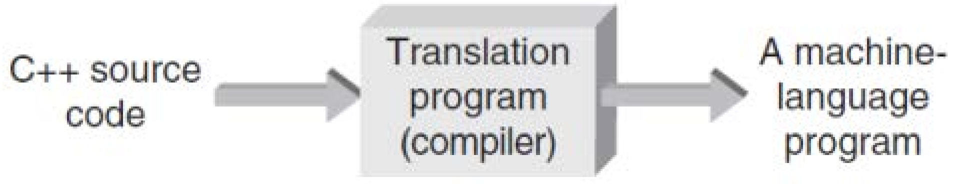
Example of Flowchart

- Question:
  - How do you design a flowchart to calculate the average of three numbers?



Program Translation

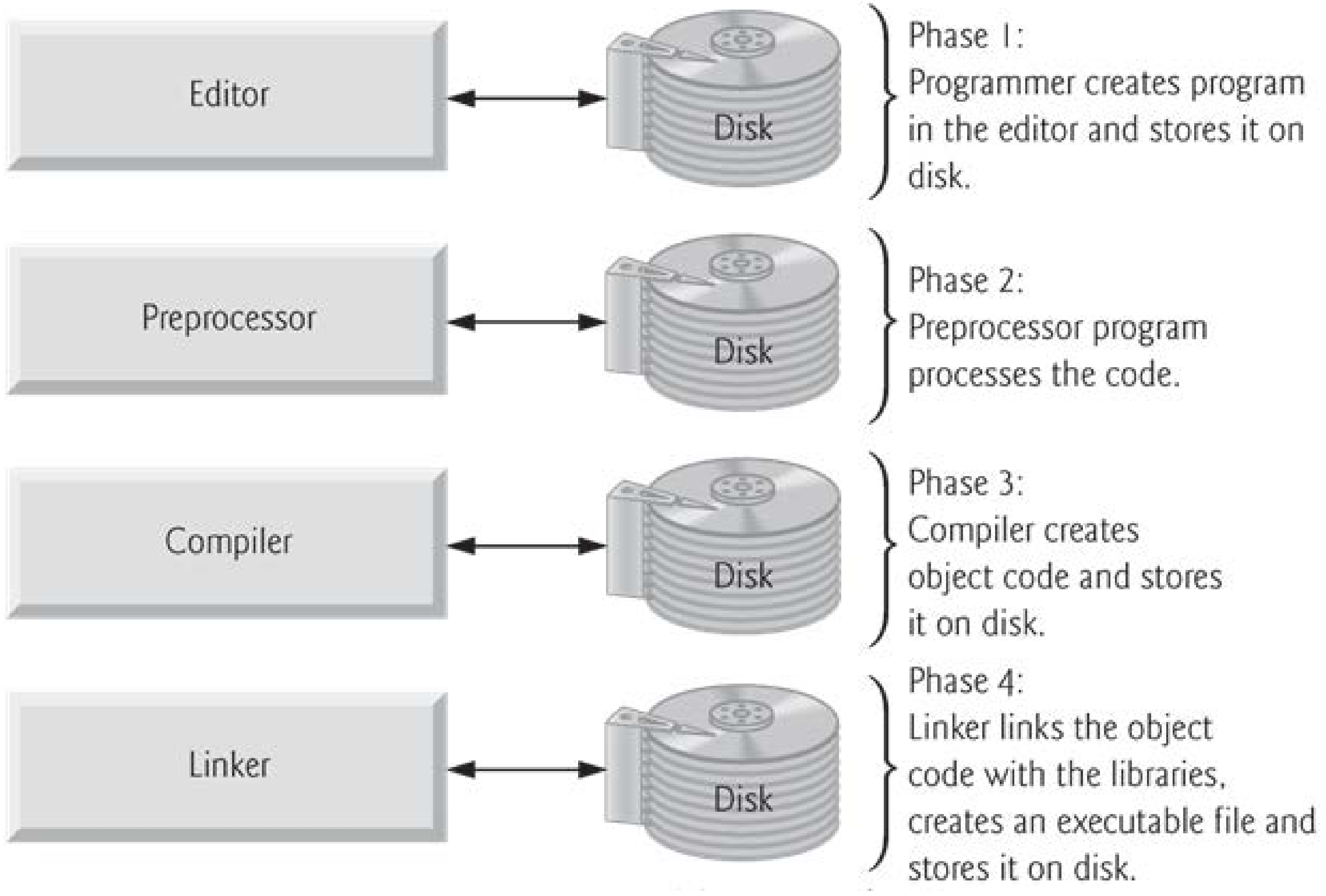
- C source program
  - Set of instructions written in C language
- Machine language
  - Internal computer language
  - Consists of a series of 1s and 0s
- Source program cannot be executed until it is translated into machine language
  - Interpreted language translates one statement at a time
  - Compiled language translates all statements together



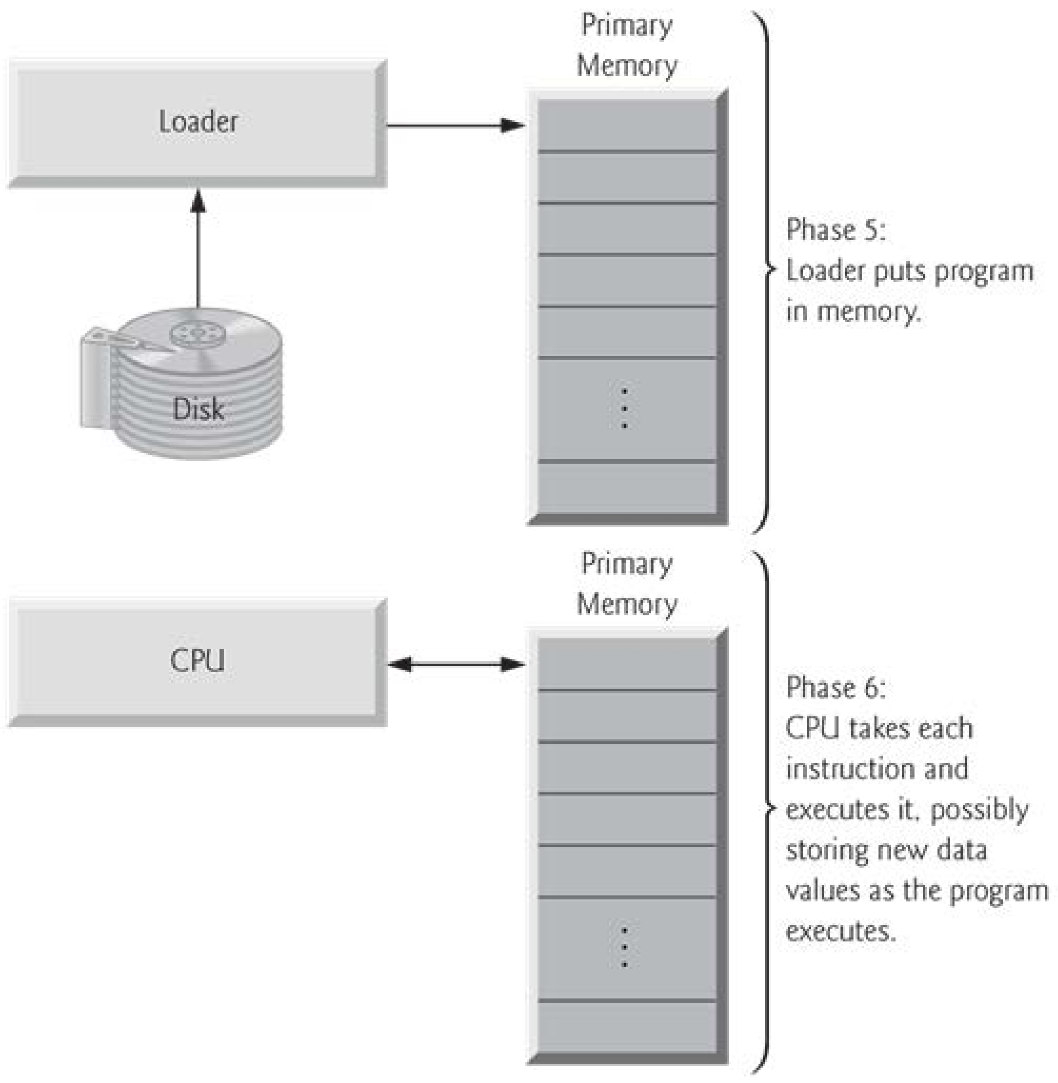
Typical C/C++ Development Environment

- C/C++ systems generally consist of three parts: a program development environment, the language and the C/C++ Standard Library
- C/C++ programs typically go through six phases: edit, preprocess, compile, link, load and execute

Typical C/C++ Development Environment (cont.)


Editor
Disk
Phase 1:
Programmer creates program in the editor and stores it on disk.
Preprocessor
Disk
Phase 2:
Preprocessor program processes the code.
Compiler
Disk
Phase 3:
Compiler creates object code and stores it on disk.
Linker
Disk
Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.

Typical C/C++ Development Environment (cont.)


Loader
Disk
Primary Memory
Phase 5:
Loader puts program in memory.
CPU
Primary Memory
Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes.

Phase 1: Creating a Program in an Editor

- Type a C/C++ program (source code) using the editor.
- C/C++ source code filenames
  - often end with the .c/.cpp extensions
- Two editors widely used on UNIX systems
  - vi (vim) and emacs
- C/C++ software packages (which has editors integrated in the programming environment) in Windows systems
  - Microsoft Visual C++
  - Dev-C++
  - Code::Blocks

Phases 2 and 3: Preprocessing and Compiling a C/C++ Program

- In phase 2, you give the command to compile the program
  - In Linux, we use gcc/g++ command to compile the c/c++ program
- In a C/C++ system, a preprocessor program executes automatically before the compiler's translation phase begins
- The C/C++ preprocessor obeys commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation
- In phase 3, the compiler translates the C/C++ program into machine-language code (also referred to as object code)

Phrase 4: Linking

- The object code produced by the C/C++ compiler typically contains “holes” due to missing parts, such as references to functions from standard libraries
- A linker links the object code with the code for the missing functions to produce an executable program

Phrase 5 and 6: Loading and Executing a Program

- Before a program can be executed, it must first be placed in memory
- This is done by the loader, which takes the executable image from disk and transfers it to memory
- Additional components from shared libraries that support the program are also loaded

- Finally, in Phase 6, the computer, under the control of its CPU, executes the program one instruction at a time

First Program in C

First Program in C: Printing a Line of Text

- source code:

```
/* Fig. 2.1: fig02_01.c
   A first program in C     */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome to C!\n" );

    return 0; /*indicate the program ended successfully*/
} /* end function main */
```

comments

preprocessor directive

Main function

- screen output:

```
Welcome to C!
```

First Program in C++: Printing a Line of Text

- source code:

```
// Fig. 2.1: fig02_01.cpp
// Text-printing program.
#include <iostream>

// function main begins program execution
int main()
{
    std::cout << "Welcome to C++!\n"; //display message

    return 0; // indicate the program ended successfully
} // end function main
```

comments

preprocessor directive

Main function

- screen output:

```
Welcome to C++!
```

Comment

- Explanatory remarks written within program
  - Clarify purpose of the program
  - Describe objective of a group of statements
  - Explain function of a single line of code
- Computer ignores all comments
  - Comments exist only for convenience of reader
- A well-constructed program should be readable and understandable
  - Comments help explain unclear components

Comment (cont.)

- Block comment
  - Multiple-line comment begins with the symbols `/*` and ends with the symbols `*/`

```
/* This is a block comment that
   spans
   three lines */
```

- Line comment
  - Begins with two slashes(`//`) and continues to the end of the line
  - Can be written on line by itself or at the end of line that contains program code

```
// this is a line comment
```

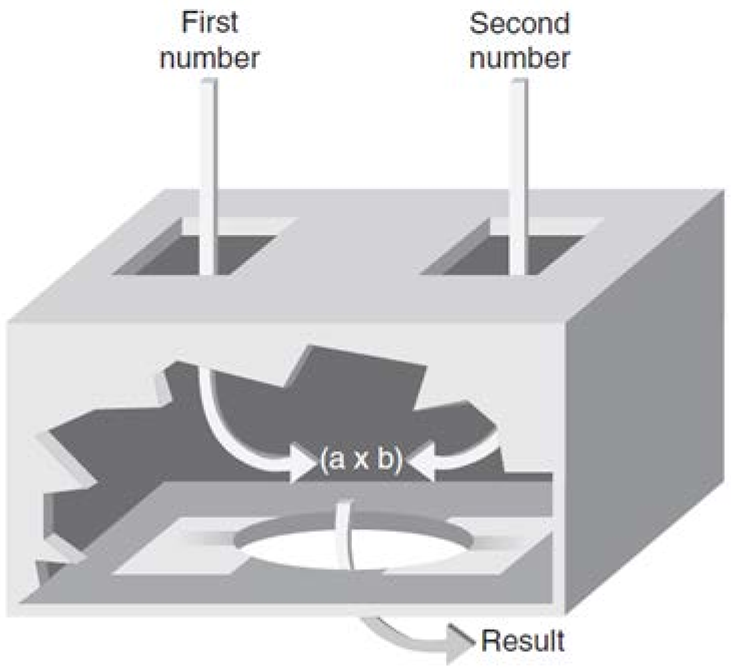
Preprocessor Directive

- A preprocessor directive is a message to the C preprocessor.
- Lines that begin with `#` are processed by the preprocessor before the program is compiled.

- `#include <stdio.h>` notifies the preprocessor to include in the program the contents of the sttandard input/output header file `<stdio.h>`.
  - Must be included for any program that outputs data to the screen or inputs data from the keyboard using C-style input/output.

Function

- Function: Name of a C/C++ procedure
  - Composed of sequence of C/C++ instructions
  - Function interface is its inputs and results
  - Method of converting input to results is encapsulated and hidden within function

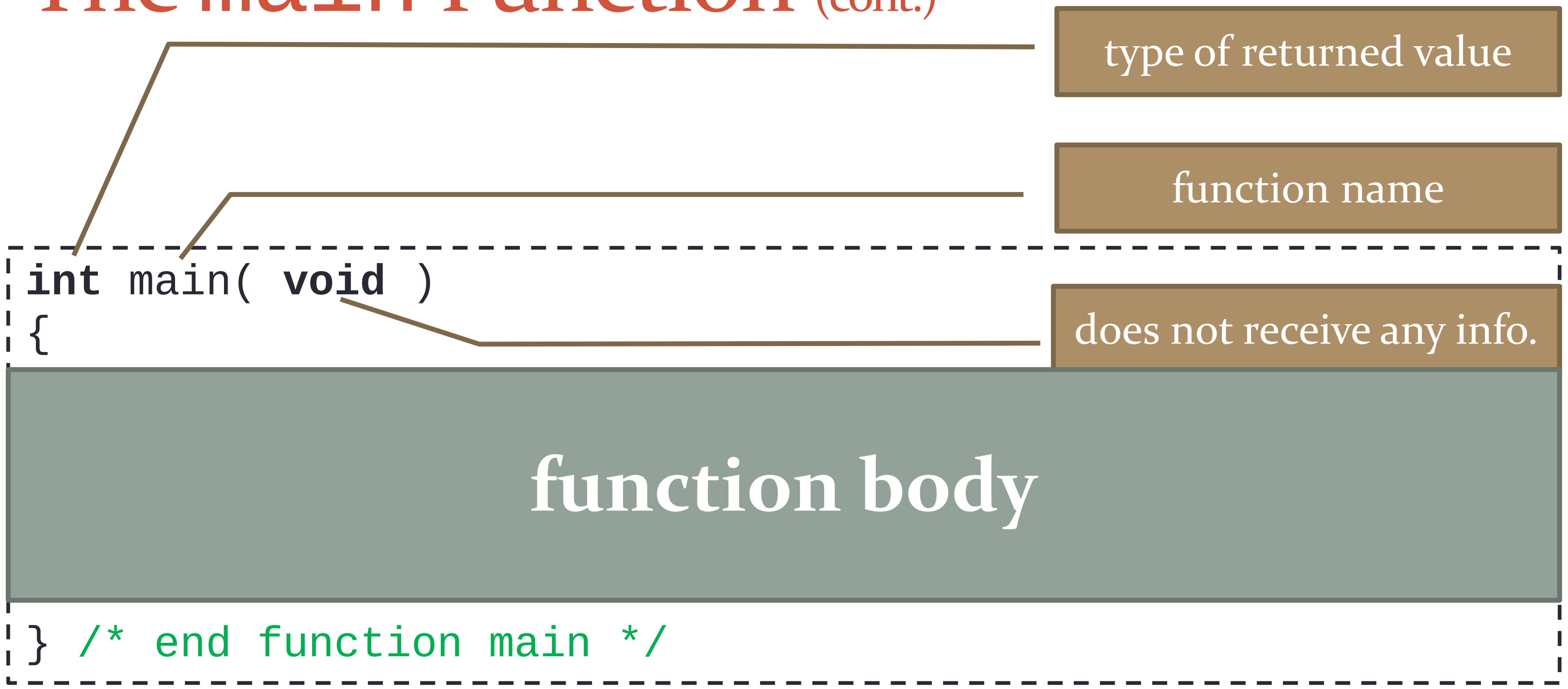


The `main` Function

- Each C/C++ program must have one and only one function named `main`
- Called a driver function because it drives the other modules

- First line of function is called header line
  - What type of data, if any, is returned from function
  - The name of function
  - What type of data, if any, is sent into function
- Data transmitted into function at runtime are referred to as arguments of function

The `main` Function (cont.)



```
int main( void )
{

} /* end function main */
```

- The keyword **`int`** to the left of `main` indicates that `main` “returns” an integer value.
- A keyword is a word in code that is reserved by C/C++ for a specific use.

Function Body: Statements

- Function body consists of a lot of statements
- A statement instructs the computer to perform an action
- A statement normally ends with a semicolon (`;`), also known as the statement terminator
  - Preprocessor directives (like `#include`) do not end with a semicolon

```
printf( "Welcome to C!\n" );
```

Printing Strings

- The `printf` function displays a string literal - characters enclosed in double quotation marks - it doesn't show the quotation marks
- `printf` doesn't automatically advance to the next output line when it finishes printing
- To make `printf` advance one line, include `\n` (the new-line character) in the string to be printed
- The new-line character can appear more than once in a string literal:

```
printf( "Welcome to C!\nHappy New Year!\n " );
```

Newline Escape Sequence

- The characters `\n` are not printed on the screen
- The backslash (`\`) is called an escape character
  - It indicates that a "special" character is to be output
- When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an escape sequence
- The escape sequence `\n` means newline
  - Causes the cursor to move to the beginning of the next line on the screen

Example of Escape Sequences

Escape Sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash.
\’	Single quote.
\”	Double quote

Existing from a Function

- When the `return` statement is used at the end of `main` the value 0 indicates that the program has terminated successfully

Modifying Our First C Program: Printing a Line of Text

- source code:

```
/* Fig. 2.3: fig02_03.c
   Printing on one line with two printf statements */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome " );
    printf( "to C!\n" );
    return 0; /*indicate the program ended successfully*/
} /* end function main */
```

- screen output:

```
Welcome to C!
```

Modifying Our First C Program: Printing Multiple Lines of Text

- source code:

```
/* Fig. 2.4: fig02_04.c
   Printing multiple lines of text with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome\nto\nC++!" );

    return 0; /*indicate the program ended successfully*/
} /* end function main */
```

- screen output:

```
Welcome
to
C++!
```

Another C Program: Adding Integers

```
/* Fig. 2.5: fig02_05.c
   Addition program        */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    /* variable declarations
    int integer1; /* first integer to add */
    int integer2; /* second integer to add */
    int sum;      /* sum of integer1 and integer2 */

    printf( "Enter first integer\n" ); /* prompt */
    /* read an integer */
    scanf( "%d", &integer1 );
```

declaration

Another C++ Program: Adding Integers (cont.)

```
    printf( "Enter second integer\n" ); /* prompt */
    /* read an integer */
    scanf( "%d", &integer2 );

    /* add the integers; store result in sum */
    sum = integer1 + integer2;

    /* display sum; end line */
    printf( "Sum is %d\n", sum);

    return 0; /*indicate the program ended successfully*/
} /* end function main */
```

Another C++ Program: Adding Integers (cont.)

- screen output:

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Declaration Statement

- To name a variable and specify the data type that can be stored in it, you use a *declaration statement*

```
int integer1;
int integer2;
int sum;
```

```
<declaration syntax>:
dataType   variableName ;
```

- The identifiers `integer1`, `integer2` and `sum` are the names of variables
- A variable is a location in the computer’s memory where a value can be stored for use by a program

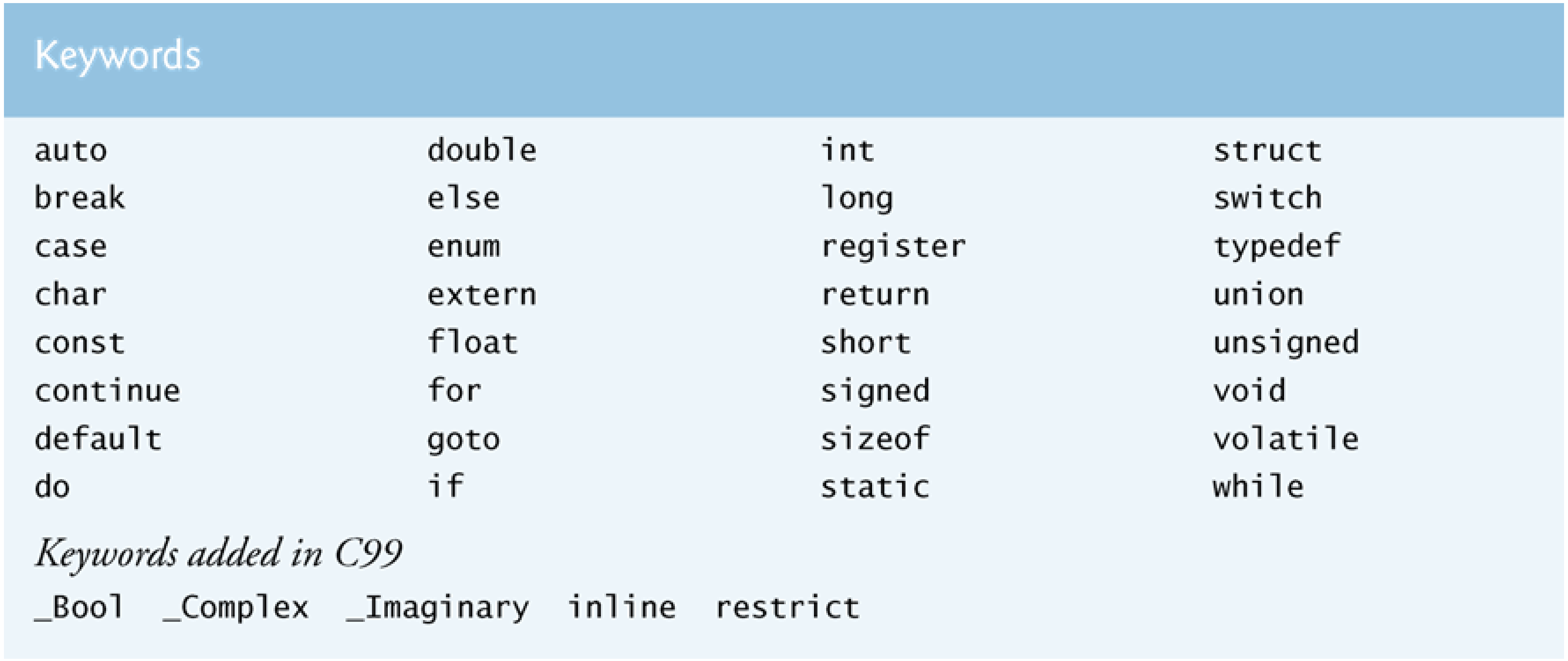
Identifier

- Identifiers are the names you supply for variables, types, functions, and labels in your program
- Identifier composition rules:
  - First character must be a letter or underscore (_)
  - Only letters, digits or underscores can follow the first letter
  - Cannon be keyword
  - Should be mnemonic

- C/C++ identifier is case-sensitive
  - Uppercase and lowercase letters are different

C Keywords

- A keyword is a word the language sets aside for a special purpose and can be used only in a specified manner.



Keywords			
`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Keywords added in C99

`_Bool` `_Complex` `_Imaginary` `inline` `restrict`

C Identifier

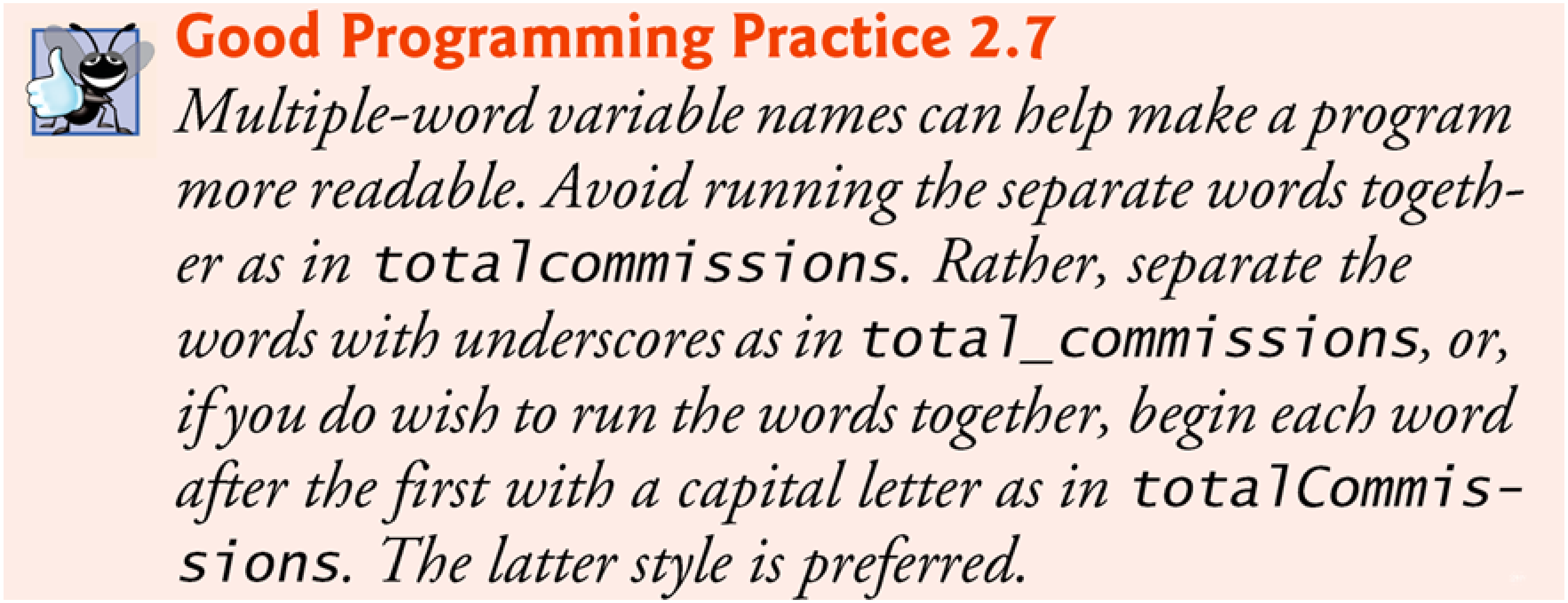
- Example of valid identifiers:

grosspay taxCalc
addNums degToRad
multByTwo salesTax
netPay bessel

Good Programming Practice 2.6

The first letter of an identifier used as a simple variable name should be a lowercase letter. Later in the text we'll assign special significance to identifiers that begin with a capital letter and to identifiers that use all capital letters.

C Identifier (cont.)



Good Programming Practice 2.7

Multiple-word variable names can help make a program more readable. Avoid running the separate words together as in `totalcommissions`. *Rather, separate the words with underscores as in* `total_commissions`, *or, if you do wish to run the words together, begin each word after the first with a capital letter as in* `totalCommissions`. *The latter style is preferred.*

C Identifier (cont.)

- Examples of invalid identifiers:

4ab3	(begins with a number)
e*6	(contains a special character)
while	(is a keyword)

Data Type

- The objective of all programs is to process data
- Data is classified into specific types
  - Numerical
  - Alphabetical
  - Audio
  - Video
- C allows only certain operations to be performed on certain types of data
  - Prevents inappropriate programming operations

Data Type (cont.)

- Data type
  - Set of values and operations that can be applied to these values
- Example of a data type: Integer
  - The values: set of all Integer (whole) numbers
    - Ex: 23, -5, 0 and 31932
  - The operations: familiar mathematical and comparison operators
    - Ex: +, -, > , <

Data Type (cont.)

- Built-In data type: Provided as an integrated part of C
  - Also known as a primitive type
  - Requires no external code
  - Consists of basic numerical types (e.g. `int`, `float`)
  - Majority of operations are symbols (e.g. `+`,`-`,`*`,…)

`int` Data Type

```
int sum;
```

- The declaration specifies that the variable `sum` is data of type `int`, meaning that the variable will hold integer values
- Examples of `int`:
  - Valid: 0 5 -10 +25 1000 253 -26351 +36
  - Invalid: $255.62 2,523 3. 6,243,982 1,492.89
- Different compilers have different internal limits on the largest and smallest values that can be stored in each data type
  - Most common allocation for `int` is four bytes

`char` Data Type

- `char` data type
  - Used to store single characters
    - Letters of the alphabet (upper- and lowercase)
    - Digits 0 through 9
    - Special symbols such as + $. , - !
- Single character value: letter, digit, or special character enclosed in single quotes
  - Examples ‘A’ ‘$’ ‘b’ ‘7’ ‘y’ ‘!’ ‘M’ ‘q’

char Data Type (cont.)

- Character values stored in ASCII or Unicode codes
- ASCII: American Standard Code for Information Exchange
  - Provides English language-based character set plus codes for printer and display control
  - Each character code contained in one byte
  - 256 distinct codes (see Appendix B: ASCII Character Set)
- Unicode: provides other language character sets
  - Each character contained in two bytes
  - Can represent 65,536 characters
  - First 256 Unicode codes have same numerical value as the 256 ASCII codes

Determining Storage Size

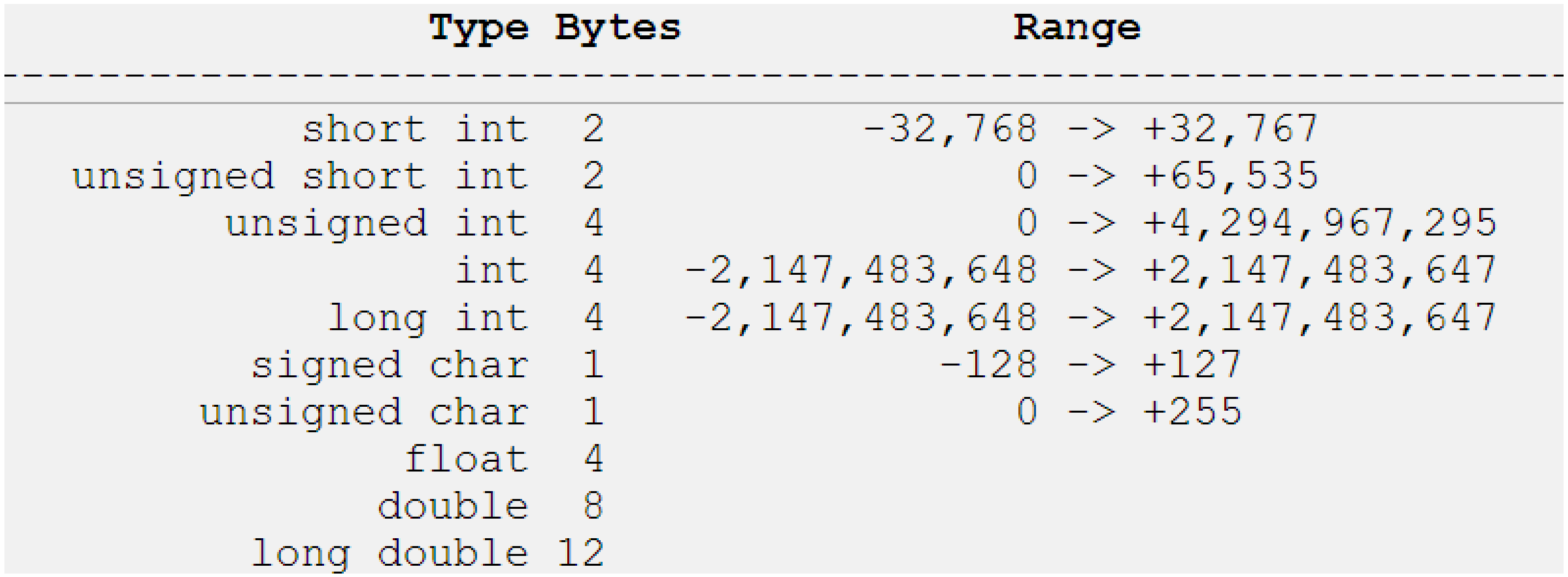
- C makes it possible to see how values are stored
- `sizeof()`
  - Provides the number of bytes required to store a value for any data type
  - Built-in operator that does not use an arithmetic symbol

```
printf("the size of int is %d\n", sizeof(int));
```

Determining Storage Size (cont.)

- Signed data type: stores negative, positive, and zero values
- Unsigned data type: stores positive and zero values
  - Provides a range of positive values double that of unsigned counterparts
- Some applications only use unsigned data types
  - Example: date applications in form *yearmonthday* (2012/10/01 => 20121001)

Determining Storage Size (cont.)



```
                Type Bytes                     Range
-------------------------------------------------------------
          short int   2             -32,768 -> +32,767
 unsigned short int   2                   0 -> +65,535
       unsigned int   4                   0 -> +4,294,967,295
                int   4      -2,147,483,648 -> +2,147,483,647
           long int   4      -2,147,483,648 -> +2,147,483,647
        signed char   1                -128 -> +127
      unsigned char   1                   0 -> +255
              float   4
             double   8
        long double  12
```

Floating-Point Types

- The number zero or any positive or negative number that contains a decimal point
  - Also called real number
  - Examples: +10.625 5.0 -6.2 3521.92 0.0
  - 5.0 and 0.0 are floating-point, but same values without a decimal (5, 0) would be integers
- C supports three floating-point types:
  - `float`: 4-bytes storage
  - `double`: 8-bytes storage
  - `long double`: 12-bytes storage

Floating-Point Types (cont.)

- Most compilers use twice the amount of storage for doubles as for floats
  - Allows a `double` to have approximately twice the precision of a `float`
- A float value is sometimes referred to as a single-precision number
- A double value is sometimes referred to as a double-precision number

Floating-Point Types (cont.)

- Precision: it can refers to either the accuracy or the number of significant digits
  - A computer programming definition
  - Significant digits = number of correct digits + 1
  - Example: if 687.45678921 has five significant digits, it is only accurate to the value 687.46

Declaration Statement

- Several variables of the same type may be declared in one declaration.

```
int number1;
int number2;
int sum;
```

```
int number1, number2, sum;
```

Declaration Statement (cont.)

- Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program

```
printf( "Enter first integer\n" ); /* prompt */
/* variable declarations */
int integer1; /* first integer to add */
/* read first integer from user into interger1 */
scanf( "%d", &integer1 );
```

Declaration Statement (cont.)

- Initialization: using a declaration statement to store a value in a variable
  - Good programming practice is to declare each initialized variable on a line by itself
  - Example: `int sum = 0;`

Reading Input

- `scanf` is the C library's counterpart to `printf`
- `scanf` requires a format string to specify the appearance of the input data
- Example of using `scanf` to read an **`int`** value:

```
/* read an integer; stores into integer1 */
scanf( "%d", &integer1 );
```

- The `&` symbol is usually (but not always) required when using `scanf`

Reading Input (cont.)

- Reading a `float` value requires a slightly different call of `scanf`:

```
/* read an float; stores into x */
scanf("%f", &x);
```

- "`%f`" tells `scanf` to look for an input value in `float` format (the number may contain a decimal point, but doesn't have to)

Assignment Statement

```
sum = integer1 + integer2;
```

- The statement adds the values of variable s `integer1` and `integer2` and assigns the results to variable `sum` using the assignment operator =
- The = operator and the + operator are called binary operators because each has two operands

Printing the Value of a Variable

- `printf` can be used to display the current value of a variable

```
printf( "Sum is %d\n", sum);
```

  - The first argument is the format control string
    - It contains the conversion specifier `%d` indicating that an integer will be printed
  - The second argument specifies the value to be printed

Printing the Value of a Variable (cont.)

- `%d` works only for **`int`** variables; to print a **`float`** variable, use `%f` instead
- By default, `%f` displays a number with six digits after the decimal point
- To force `%f` to display **`p`** digits after the decimal point, put `.p` between `%` and `f`
- To print the line

```
Profit: $2150.48
```

use the following call of `printf`:

```
printf("Profit: $%.2f\n", profit);
```

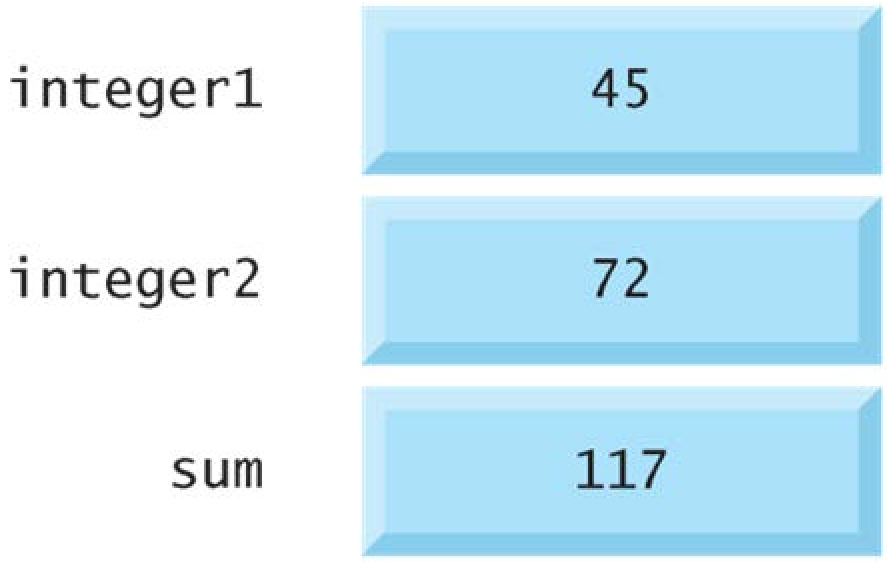
Printing the Value of a Variable (cont.)

- There's no limit to the number of variables that can be printed by a single call of `printf`:

```
printf("Height: %d Length: %d\n",
        height, length);
```

Memory Concepts

- A variable is a location in the computer's memory where a value can be stored for use by a program
- Every variable has a name, a type, a size, and a value



Memory Allocation

- Each data type has its own storage requirements
  - Computer must know variable's data type to allocate storage

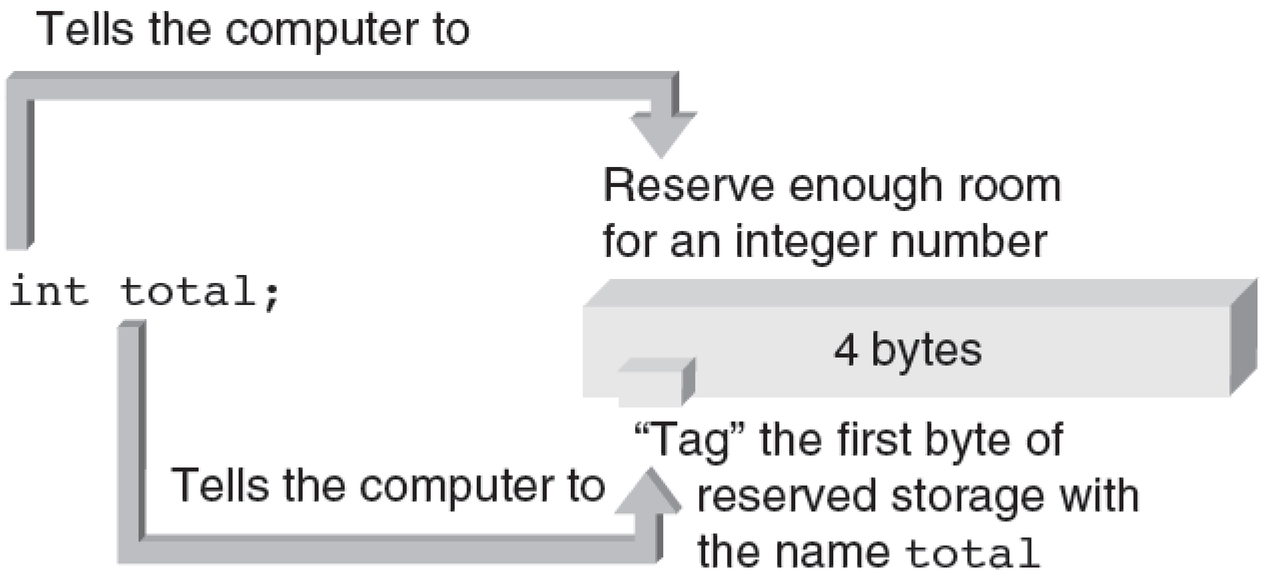

Tells the computer to
Reserve enough room for an integer number
int total;
4 bytes
Tells the computer to
"Tag" the first byte of reserved storage with the name total

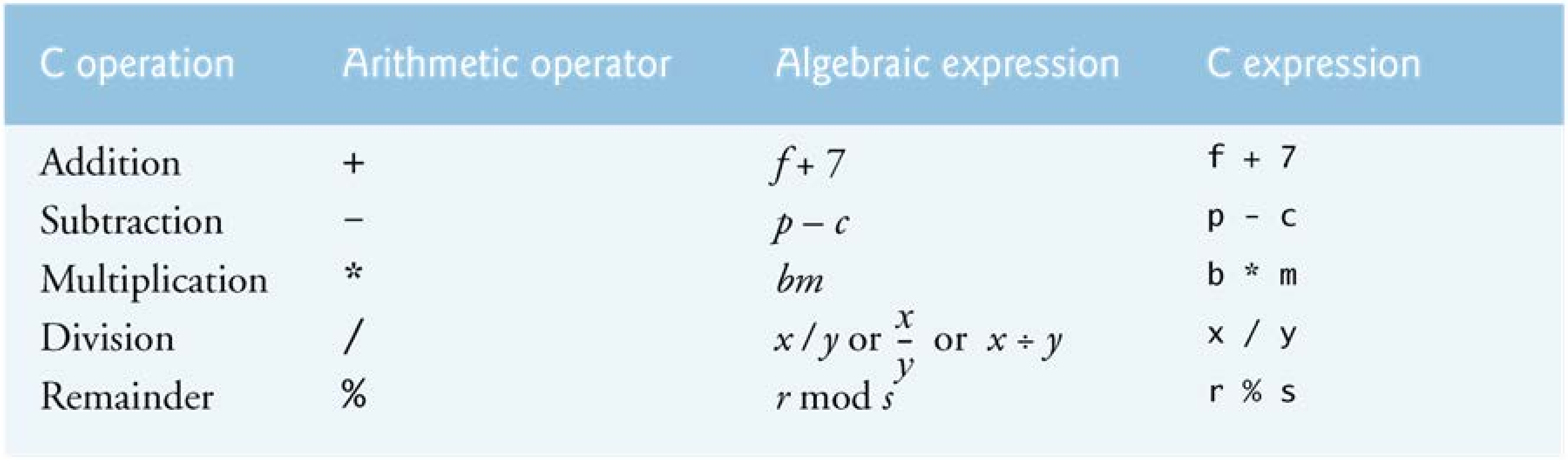
Figure 2.6a Defining the integer variable named `total`

Source: G. Bronson, "A First Book of C++," 4th Ed

Arithmetic Operators

- Arithmetic expressions contains
  - Operator: ex: +, `-`, etc…
  - Operand: including unknown variables and know numbers
- Standard arithmetic operators: +, `-`, `*`, `/`, %
  - Unary operator: the operator only requires one operand.
    - Ex: negation(`-`): `-5`
  - Binary operator: the operator requires two operands.
    - Ex: subtract(`-`): `x - y`

Arithmetic Operators (cont.)



C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Integer Division

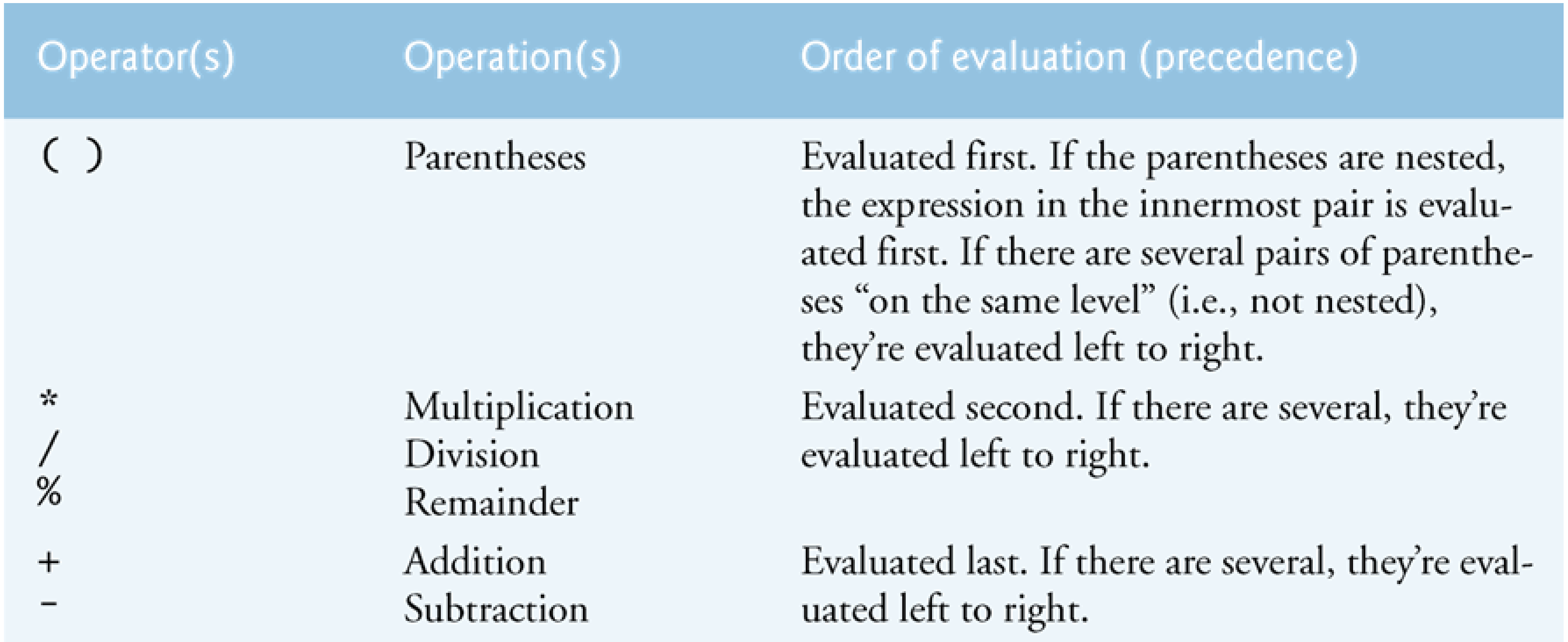
- Division of two integers yields an integer
  - Integers cannot contain a fractional part - results may seem strange
  - Example: integer 15 divided by integer 2 yields the integer result 7
- Remainder Operator (%): captures the remainder
  - Also called the remainder operator
  - Example: 9 % 4 is 1 (remainder of 9/4 is 1)

Multiple Operators Expression

- Rules for expressions with multiple operators
  - Two binary operators cannot be placed side by side
    - E.g. 5*%6 is invalid
  - Parentheses may be used to form groupings
  - Expressions within parentheses are evaluated first
  - Sets of parentheses may be enclosed by other parentheses
  - Parentheses cannot be used to indicate multiplication (multiplication operator * must be used)
    - E.g. (3+4)(4+5) is invalid

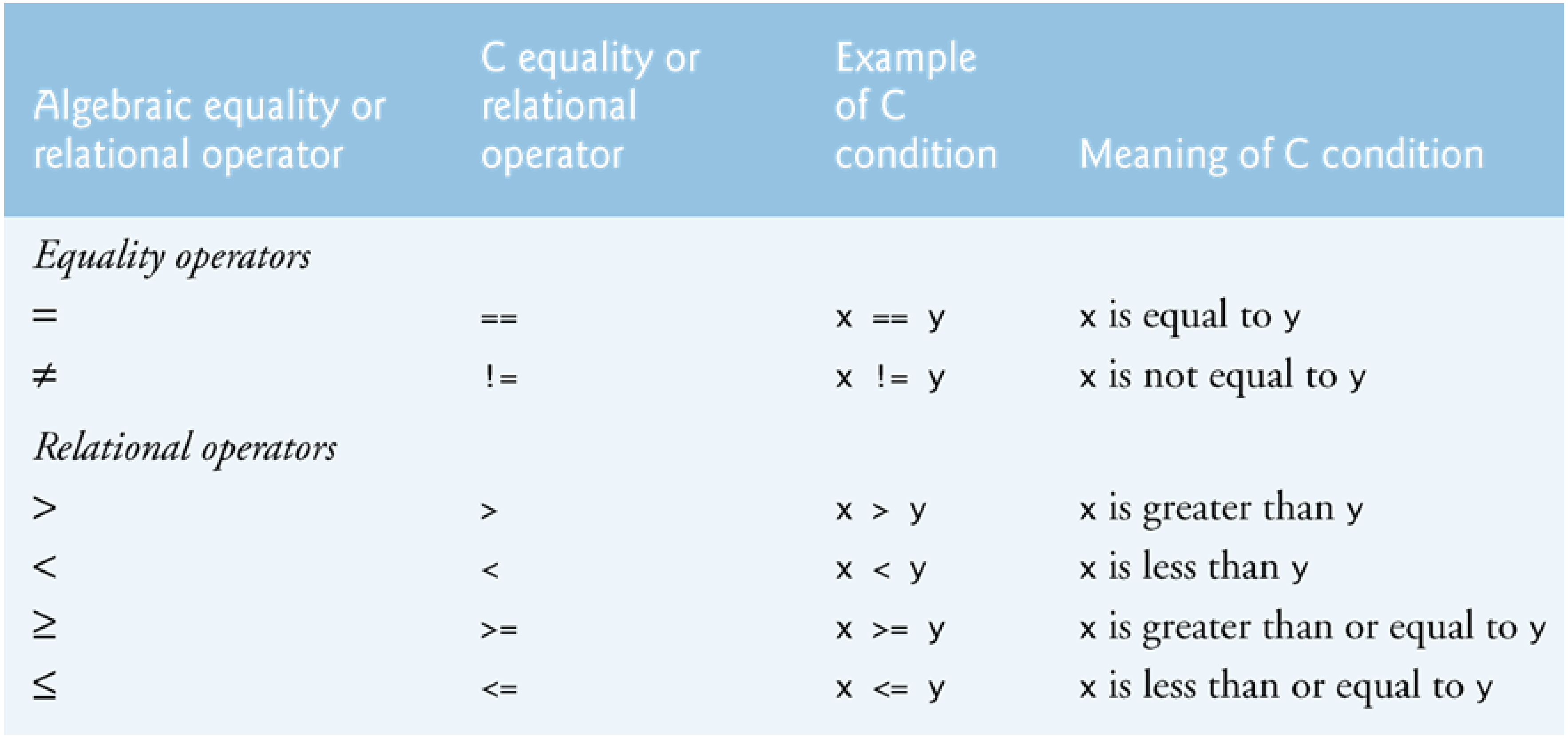
Precedence Rules

- C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those followed in algebra.



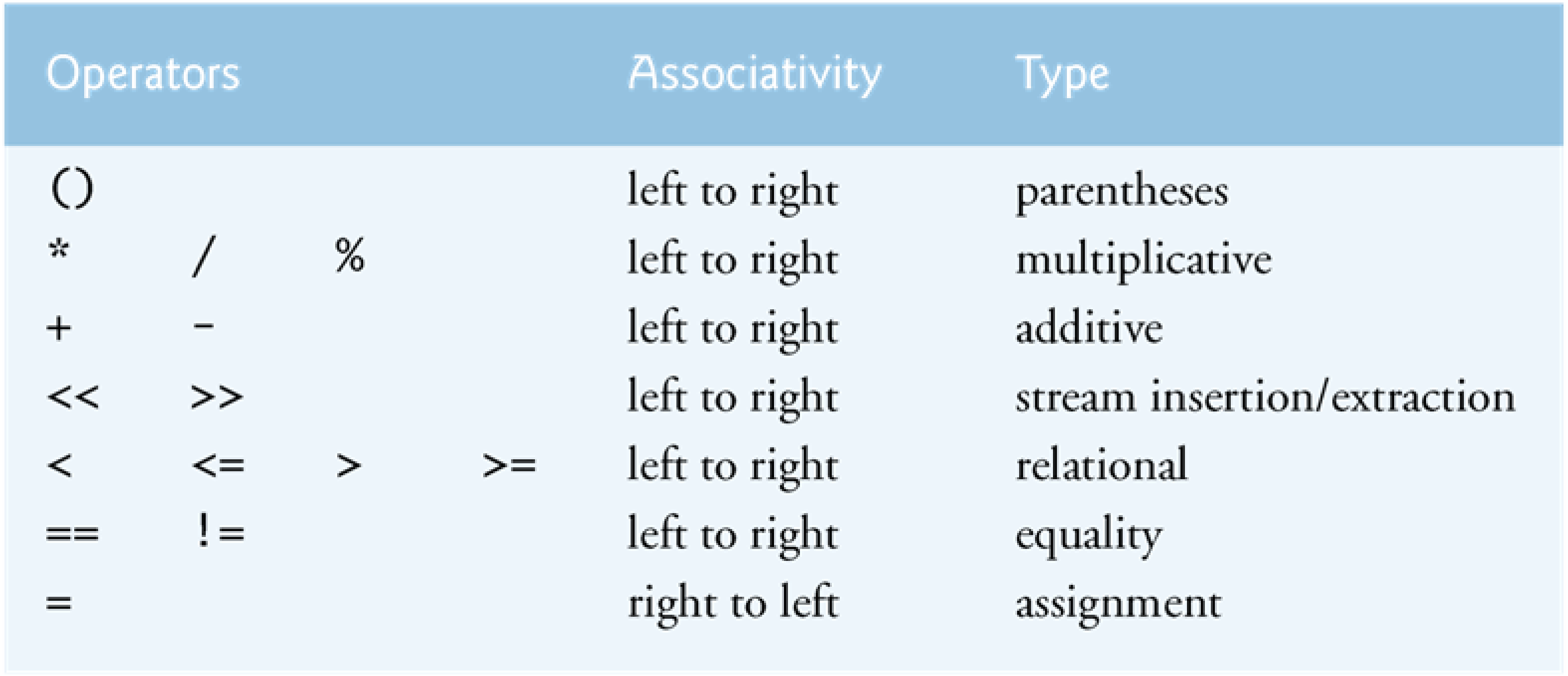
Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Equality and Relational Operators



Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y
Relational operators			
>	`>`	`x > y`	x is greater than y
<	`<`	`x < y`	x is less than y
≥	`>=`	`x >= y`	x is greater than or equal to y
≤	`<=`	`x <= y`	x is less than or equal to y

Precedence and Associativity Rules



Operators	Associativity	Type
`()`	left to right	parentheses
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<<` `>>`	left to right	stream insertion/extraction
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`=`	right to left	assignment

The Decision Making Program

```
// Fig. 2.13: fig02_13.c (partial codes)
#include <stdio.h>

// function main begins program execution
int main()
{
    int num1; // first integer to compare
    int num2; // second integer to compare

    printf("Enter two integers, and I will tell you\n");
    printf("the relationships they satisfy: ");

    scanf("%d%d", &num1, &num2); // read two integers

    if ( num1 == num2)
        printf("%d is equal to %d\n", num1, num2);

    return 0; // indicate the program ended successfully
} // end function main
```

The Decision Making Program (cont.)

- screen output

```
Enter two integers to compare: 7 7
7 == 7
```

Style of C/C++ Programs

Style of C/C++ Programs

- Every C/C++ program must contain one and only one `main()` function.
  - Statements included within braces `{}`
- C/C++ allows flexibility in format for the word `main`, the parentheses `()`, and braces `{}`
  - More than one statement can be put on line
  - One statement can be written across lines
- Use formatting for clarity and ease of program reading

Standard C/C++ Program Form

- Function name starts in column 1
  - Name and parentheses on their own line
- Opening brace of function body is on next line
  - Aligned with first letter of function name
- Closing brace is on last line of function
  - Aligned with opening brace
- Standard form highlights the function as a unit

Standard C/C++ Program Form (cont.)

- Within function, indent statements 4 spaces (= 1 tab)
    - Creates uniform look for similar statement groups
    - Good programming practice
- Final program form should be consistent
    - Proper format improves program readability and understandability

Bad Program Format

```
int main
        (
        ) { first statement; second statement;
            third statement;
       fourth statement;
return 0;}
```

Good Program Format

```
int main ()
{
    first statement;
    second statement;
    third statement;
    fourth statement;

    return 0;
}
```

Common Programming Errors

- Omitting parentheses after `main`
- Omitting or incorrectly typing the opening brace(`{`)/closing brace(`}`)
- Misspelling the name of an object or function
- Forgetting to close a string sent to `printf` with a double-quote symbol
- Omitting the semicolon (`;`) at the end of each statement
- Forgetting `\n` to indicate a new line

Summary

- Typical C/C++ Development Environment
- The simplest C program has the form:

```
#include <stdio.h>

int main ( void )
{
    program statements;

    return 0;
}
```

Summary (cont.)

- C/C++ statements are terminated by a semicolon
- C standard library contains many functions
  - Standard library provided with C compiler
  - Include `<stdio.h>` for input and output
- `printf` function call displays text or numeric results to screen
- `scanf` function call gests text or numeric results from keyboard
- Identifiers in C is case-sensitive
  - Using meaningful names