Chapter 2 Introduction to C Programming

C How to Program

2.2 A Simple C Program: Printing a Line of Text

```c
// 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" );
} // end function main
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- Lines 1 and 2
  - `// Fig. 2.1: fig02_01.c`
    `A first program in C`
- begin with `//`, indicating that these two lines are comments.
  - improve program readability.
  - Comments are ignored by the C compiler .
- You can also use /*…*/ multi-line comments
  - everything from /* to */ is a comment.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

***#include* Preprocessor Directive**

- Line 3
    - `#include <stdio.h>`
- is a directive to the C preprocessor.
    - beginning with #

- `<stdio.h>`: standard input/output header
    - Program will use stdio library functions

2.2 A Simple C Program: Printing a Line of Text (Cont.)

The `main` Function

- Line 6
  - `int main( void )`
- is the entrance of every C program.
- The parentheses after `main` indicate that `main` is a program building block called a function.

```
 int  main(void)
{
     program body
}
```

2.2 A Simple C Program: Printing a Line of Text (Cont.)

An Output Statement

- Line 8
  - `printf( "Welcome to C!\n" );`
- instructs the computer to perform an action, namely to print on the screen the string of characters marked by the quotation marks.
- Every statement must end with a semicolon (;)
- `\n` means newline.
  - The backslash (\) is called an escape character.

Escape sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Produces a sound or visible alert without changing the current cursor position.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double-quote character in a string.

Fig. 2.2 | Some common escape sequences .

2.2 A Simple C Program: Printing a Line of Text (Cont.)

Using Multiple* `printf`*s

```c
// 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" );
} // end function main
```

```
Welcome to C!
```

Fig. 2.3 | Printing one line with two `printf` statements.

- One `printf` can print *several* lines by using additional newline characters
  - printf("Welcome \n to C!\n");

2.3 Another Simple C Program: Adding Two Integers

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

// function main begins program execution
int main( void )
{
   int integer1; // first number to be entered by user
   int integer2; // second number to be entered by user

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

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

   int sum; // variable in which sum will be stored
   sum = integer1 + integer2; // assign total to sum

   printf( "Sum is %d\n", sum ); // print sum
} // end function main
```

Fig. 2.5 | Addition program. (Part 1 of 2.)

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

Fig. 2.5 | Addition program. (Part 2 of 2.)

2.3 Another Simple C Program: Adding Two Integers (Cont.)

Variables and Variable Definitions

- Lines 8–10

  -
    ```
    int integer1; /* first number to be input by user */
    int integer2; /*second number to be input by user */
    int sum; /* variable in which sum will be stored */
    ```

  are definitions.

- The names `integer1`, `integer2` and `sum` are the names of variables—locations in memory where values can be stored for use by a program.

- Combine to a single statement
  - `int integer1, integer2, sum;`

2.3 Another Simple C Program: Adding Two Integers (Cont.)

Identifiers and Case Sensitivity

- A variable name in C is any valid identifier.
  - letters, digits and underscores (_) that does *not* begin with a digit.

- C is case sensitive—uppercase and lowercase letters are different in C

2.3 Another Simple C Program: Adding Two Integers (Cont.)

Prompting Messages

- Line 12
  - `printf( "Enter first integer\n" ); /* prompt */`

  displays the literal "`Enter first integer`" and positions the cursor to the beginning of the next line.
- This message is called a prompt

2.3 Another Simple C Program: Adding Two Integers (Cont.)

The `scanf` Function and Formatted Inputs

- The next statement
  - `scanf( "%d", &integer1 ); /* read an integer */`

  uses `scanf` to obtain a value from the user.
- The `scanf` function reads from the standard input, which is usually the keyboard.
- `%d` conversion specifier indicates that the data should be an integer
- (&)—called the address operator in C

2.3 Another Simple C Program: Adding Two Integers (Cont.)

Assignment Statement

- The assignment statement in line 18
    - `sum = integer1 + integer2; /* assign total to sum */`
  - = is called assignment operator

Printing with a Format Control String

- Line 20
    - `printf( "Sum is %d\n", sum ); /* print sum */`

  calls function `printf` to print the literal `Sum is` followed by the numerical value of variable `sum` on the screen.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

Calculations in `printf` Statements

- We could have combined the previous two statements into the statement
  - `printf( "Sum is %d\n", integer1 + integer2 );`

- The right brace, `}`, at line 21 indicates that the end of function `main` has been reached.

2.4 Memory Concepts

- Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer's memory.
- Every variable has a name, a type and a value.

- `scanf( "%d", &integer1 );`
  - the value entered by the user is placed into a memory location named `integer1`

- Whenever a value is placed in a memory location, the value replaces the previous value in that location

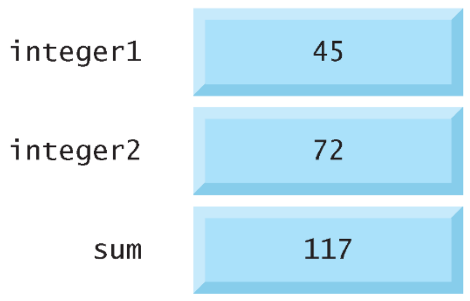
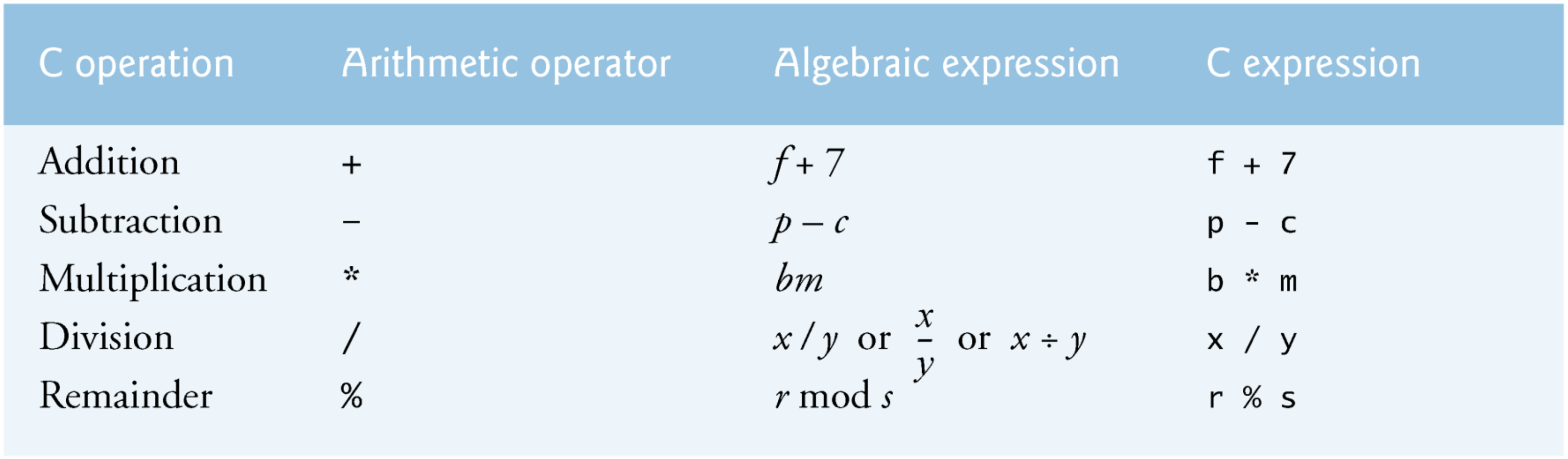


Fig. 2.8 | Memory locations after a calculation.

2.5 Arithmetic in C

- The asterisk (`*`) indicates multiplication and the percent sign (`%`) denotes the remainder operator.
- x = ab;   WRONG!
- x = a * b; CORRECT!



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`

Fig. 2.9 | Arithmetic operators.

2.5 Arithmetic in C (Cont.)

Integer Division and the Remainder Operator

- Integer division yields an integer result.
  - For example, the expression `7 / 4` evaluates to `1` and the expression `17 / 5` evaluates to `3`.
- `%`: remainder operator
  - `x % y` yields the remainder after `x` is divided by `y`
  - `7 % 4` yields `3` and `17 % 5` yields `2`.

2.5 Arithmetic in C (Cont.)

Parentheses for Grouping Subexpressions

- Parentheses are used in C expressions in the same manner as in algebraic expressions.
  - `a = a * ( b + c )`

- In cases of nested, or embedded, parentheses, such as
  - `( ( a + b ) + c )`

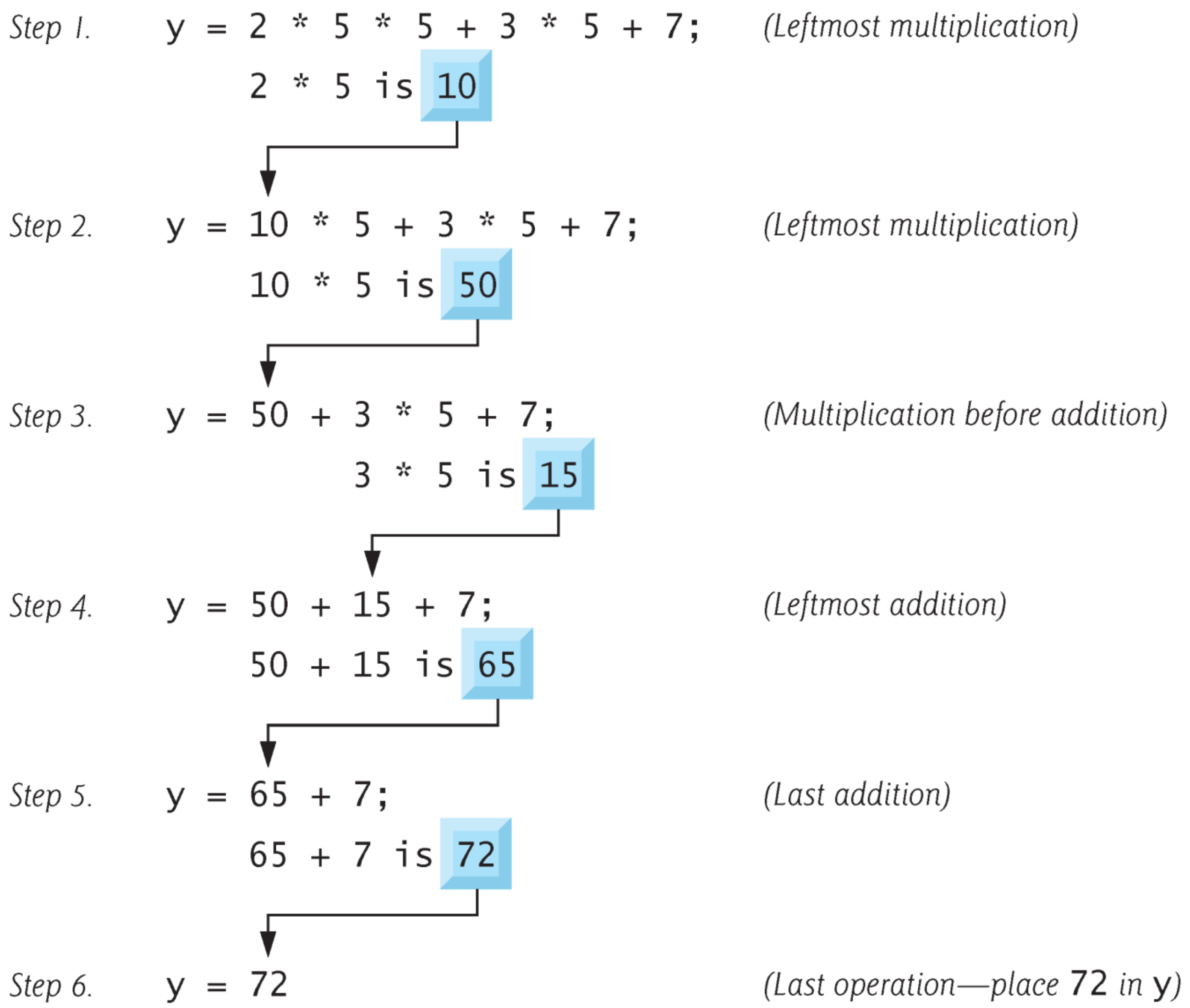
  the operators in the innermost pair of parentheses are applied first.

2.5 Arithmetic in C (Cont.)

- Operator precedence

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 third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.



Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72` *(Last operation—place* `72` *in* `y`*)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

2.6 Decision Making: Equality and Relational Operators

- C’s `if` statement that allows a program to make a decision based on the truth or falsity of a statement of fact called a condition.

- if (condition){

- }
- else {

- }

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
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
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y

Fig. 2.12 | Equality and relational operators.

```c
// Fig. 2.13: fig02_13.c
// Using if statements, relational
// operators, and equality operators.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   int num1; // first number to be read from user
   int num2; // second number to be read from user

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

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

```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 1 of 3.)

```
    if ( num1 != num2 ) {
       printf( "%d is not equal to %d\n", num1, num2 );
    } // end if

    if ( num1 < num2 ) {
       printf( "%d is less than %d\n", num1, num2 );
    } // end if

    if ( num1 > num2 ) {
       printf( "%d is greater than %d\n", num1, num2 );
    } // end if

    if ( num1 <= num2 ) {
       printf( "%d is less than or equal to %d\n", num1, num2 );
    } // end if

    if ( num1 >= num2 ) {
       printf( "%d is greater than or equal to %d\n", num1, num2 );
    } // end if
 } // end function main
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 2 of 3.)

```
Enter two integers, and I will tell you
the relationships they satisfy: **3 7**
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: **22 12**
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: **7 7**
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 3 of 3.)

Operators				Associativity
()				left to right
*	/	%		left to right
+	-			left to right
<	<=	>	>=	left to right
==	!=			left to right
=				right to left

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

2.6 Arithmetic in C (Cont.)

- `int` and `if`—are keywords or reserved words of the language.
- These words have special meaning to the C compiler, so you must be careful not to use these as identifiers such as variable names.

Keywords				
auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	
Keywords added in C99 standard				
_Bool	_Complex	_Imaginary	inline	restrict
Keywords added in C11 standard				
_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			

Fig. 2.15 | C's keywords.