CSE1142 - Introduction to C Programming

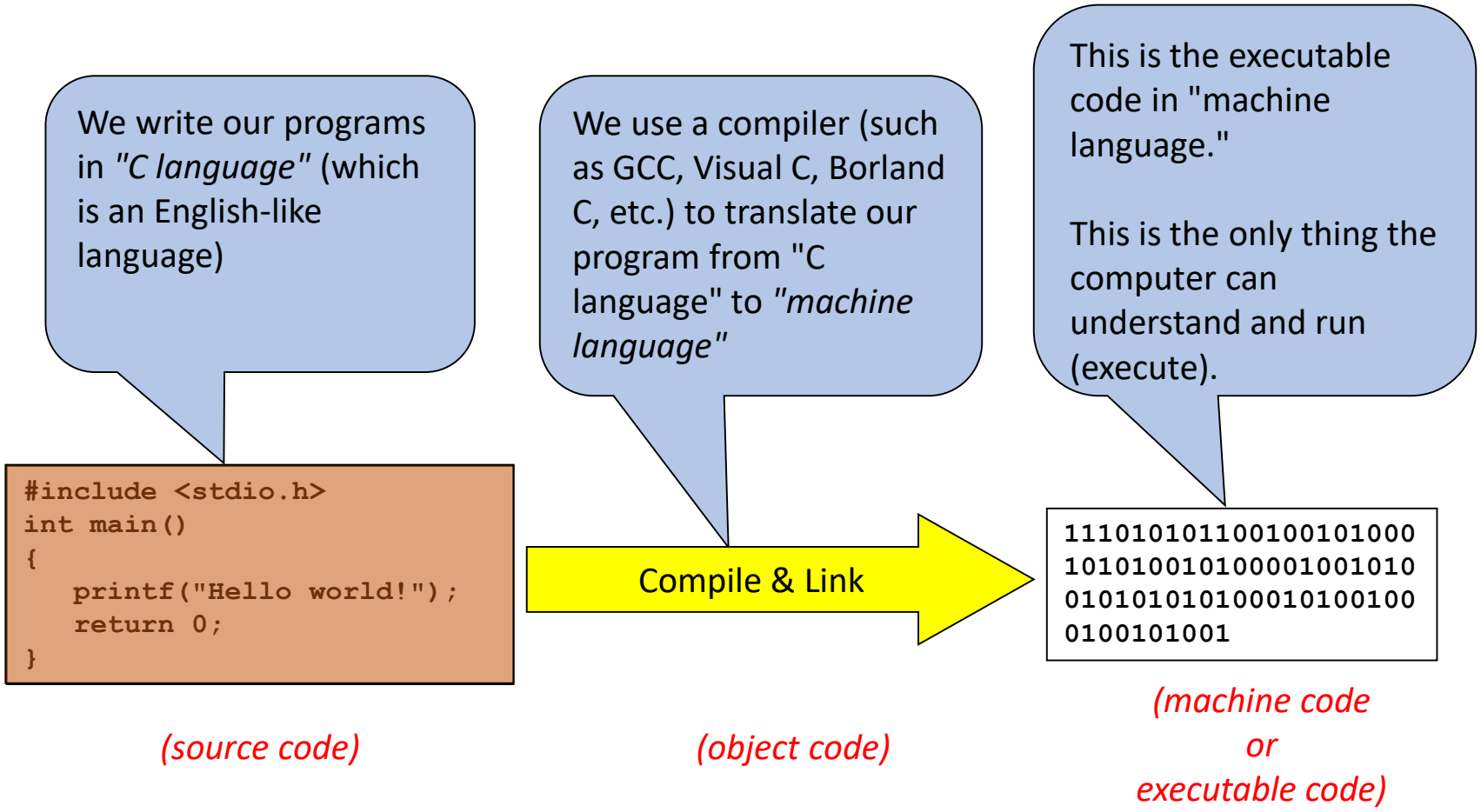
Sanem Arslan Yılmaz

Slides are adapted from:
[1] C How to Program 8th Edition, by Deitel & Deitel.
[2] Problem Solving & Program Design in C, Eighth Edition, R. Hanly & Elliot B. Koffman.

C

- A high-level programming language
- Developed in 1972 by Dennis Ritchie at AT&T Bell Labs
- Designed as language to write the Unix operating system
- Resembles everyday English
- Very popular

Welcome to C Programming Language



A Simple C Program: Printing a Line of Text

- We begin by considering a simple C program.
- Our first example prints a line of text (Fig. 2.1).

```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.

- Comments:
  - `// Fig. 2.1: fig02_01.c`
    `// A first program in C`
  - begin with `//`, indicating that these two lines are comments.
  - For multi-line comments, use `/*… */`.
- ***#include Preprocessor Directive***
  - **`#include <stdio.h>`**
  - is a directive to the C preprocessor.
  - Tells computer to load contents of a certain file.
  - `<stdio.h>` allows standard input/output operations.

[1] C How to Program 8th Edition, by Deitel & Deitel.

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

The `main` Function

- **`int main( void )`**
  - is a part of every C program.
  - The parentheses after `main` indicate that `main` is a program building block called a function.
  - C programs contain one or more functions, exactly one of which must be `main`
  - Parenthesis used to indicate a function
  - **`int`** means that `main` "returns" an integer value
  - Braces (`{` and `}`) indicate a block
    - The bodies of all functions must be contained in braces
  - The `void` in parentheses here means that `main` does not receive any information.

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

An Output Statement

- `printf( "Welcome to C!\n" );`
  - instructs the computer to perform an action,
    - print on the screen the string of characters marked by the quotation marks(“ “).
  - Escape character (\)
    - Indicates that **printf** should do something out of the ordinary
    - Notice that the characters \n were not printed on the screen.
    - \n is the newline character

- `return 0;`
  - A way to exit a function
  - `return` 0, in this case, means that the program terminated normally

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 .

```
1  // Fig. 2.3: fig02_03.c
2  // Printing on one line with two printf statements.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10 } // end function main
```

```
Welcome to C!
```

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

[1] C How to Program 8th Edition, by Deitel & Deitel.

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

// function main begins program execution
int main( void )
{
   printf( "Welcome\nto\nC!\n" );
} // end function main
```

```
Welcome
to
C!
```

Fig. 2.4 | Printing multiple lines with a single `printf`.

[1] C How to Program 8th Edition, by Deitel & Deitel.

General Form of a C Program

```
preprocessor directives
main function heading
{
    declarations
    executable statements
}
```

2.3 Another Simple C Program: Adding Two Integers

- Our next program (fig. 2.5) uses the Standard Library function `scanf` to obtain two integers typed by a user at the keyboard, computes the sum of these values and prints the result using `printf`.

```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.)

[1] C How to Program 8th Edition, by Deitel & Deitel.

```
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.)

- As before
  - Comments, `#include <stdio.h>` and `main`

Variables and Variable Definitions

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

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

The `scanf` Function and Formatted Inputs

- `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.
- This **`scanf`** statement has two arguments
  - %d - indicates data should be a decimal integer
  - `&integer1` - location in memory to store variable
  - & is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statements
- When executing the program the user responds to the **`scanf`** statement by typing in a number, then pressing the *enter* (return) key

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

Assignment Statement

- The assignment statement
  - `sum = integer1 + integer2; // assign total to sum`

  calculates the total of variables `integer1` and `integer2` and assigns the result to variable `sum` using the assignment operator =.
- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - %d means decimal integer will be printed
    - `sum` specifies what integer will be printed
  - Calculations can be performed inside `printf` statements

    ```
    printf( "Sum is %d\n", integer1 + integer2 );
    ```

The printf Function

- print list
  - in a call to printf, the variables or expressions whose values are displayed
- placeholder
  - a symbol beginning with % in a format string that indicates where to display the output value

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

Data Types

- int
  - a whole number
  - 435
- double
  - a real number with an integral part and a fractional part separated by a decimal point
  - 3.14159
- char
  - an individual character value
  - enclosed in single quotes
  - ‘A’, ‘z’, ‘2’, ‘9’, ‘*’, ‘!’

Placeholders in format string

Placeholder	Variable Type	Function Use
% c	char	printf/scanf
% d	int	printf/scanf
% f	double	printf
% lf	double	scanf

2.4 Memory Concepts

- **Variables**
  - 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.
  - In the addition program of Fig. 2.5, when the statement
    - `scanf( "%d", &integer1 ); // read an integer`
  - is executed, the value entered by the user is placed into a memory location to which the name `integer1` has been assigned.
- Suppose the user enters the number 45 as the value for `integer1`.
- The computer will place 45 into location `integer1` as shown in Fig. 2.6.

Fig. 2.6 | Memory location showing the name and value of a variable.

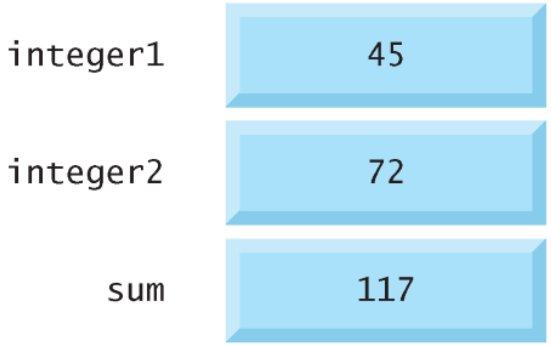


Fig. 2.8 | Memory locations after a calculation.

2.5 Arithmetic in C

- Arithmetic calculations **(+, -, *, /, %)**
  - The asterisk (*) indicates multiplication and the percent sign (%) denotes the remainder operator.
- ***Integer Division***
  - 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 %
  - Modulus operator (%) returns the remainder
  - 7 % 4 yields 3 and 17 % 5 yields 2.

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.

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	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated third. If there are several, they're evaluated left to right.
-	Subtraction	
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.

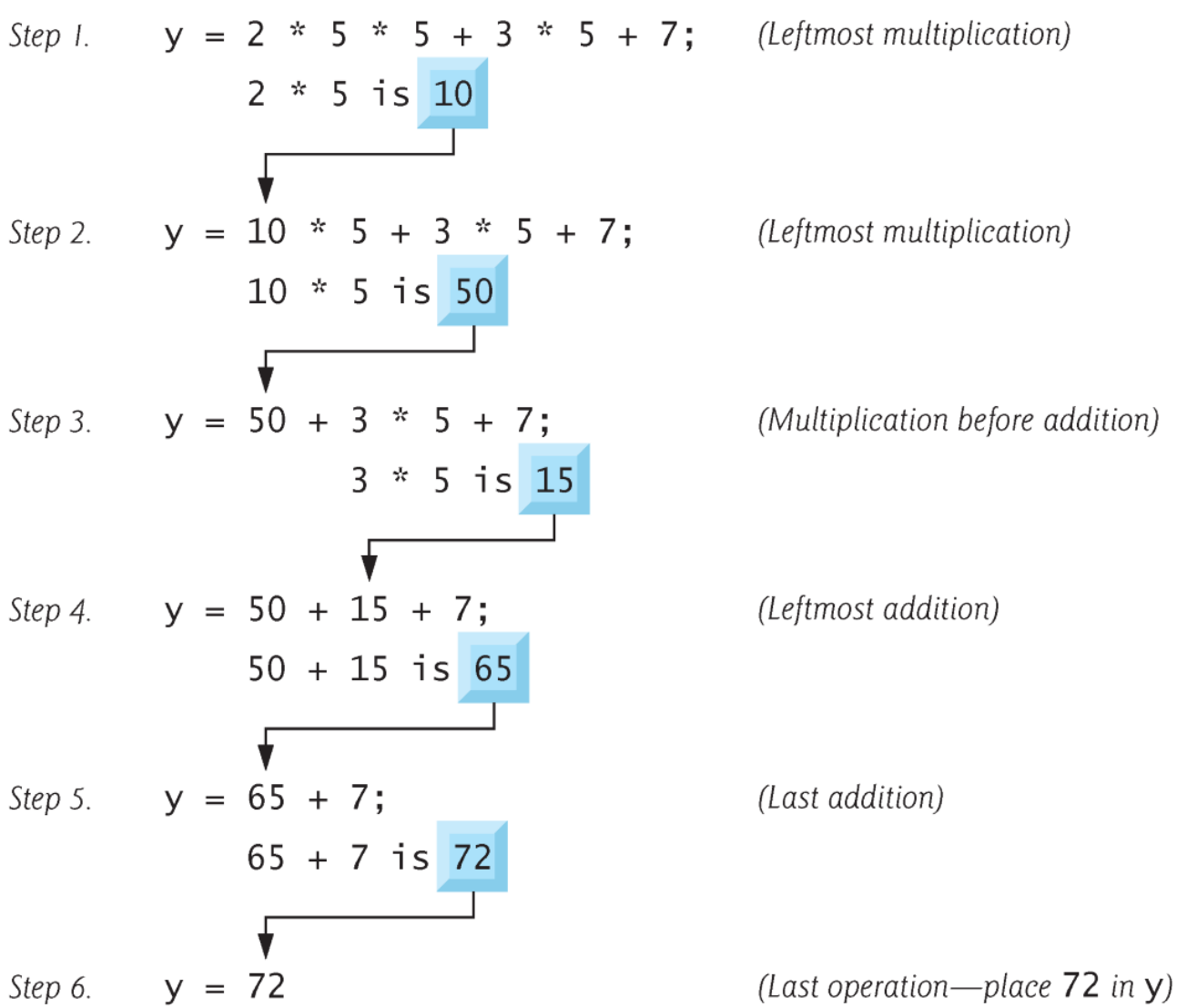


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

2.6 Decision Making: Equality and Relational Operators

- Executable C statements
  - perform actions (such as calculations or input or output of data)
  - make decisions
    - May want to print `"pass"` or `"fail"` given the value of a test grade
- `if` statement
  - allows a program to make a decision based on the truth or falsity of a statement of fact called a condition.
  - ***0*** is ***false***, ***non-zero*** is ***true***

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.

2.6 Decision Making: Equality and Relational Operators

- Figure 2.13 uses six `if` statements to compare two numbers entered by the user.
- If the condition in any of these `if` statements is true, the `printf` statement associated with that `if` executes.

```
1  // Fig. 2.13: fig02_13.c
2  // Using if statements, relational
3  // operators, and equality operators.
4  #include <stdio.h>
5
6  // function main begins program execution
7  int main( void )
8  {
9     printf( "Enter two integers, and I will tell you\n" );
10    printf( "the relationships they satisfy: " );
11
12    int num1; // first number to be read from user
13    int num2; // second number to be read from user
14
15    scanf( "%d %d", &num1, &num2 ); // read two integers
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n", num1, num2 );
19    } // end if
20
```

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

[1] C How to Program 8th Edition, by Deitel & Deitel.

```
21      if ( num1 != num2 ) {
22         printf( "%d is not equal to %d\n", num1, num2 );
23      } // end if
24
25      if ( num1 < num2 ) {
26         printf( "%d is less than %d\n", num1, num2 );
27      } // end if
28
29      if ( num1 > num2 ) {
30         printf( "%d is greater than %d\n", num1, num2 );
31      } // end if
32
33      if ( num1 <= num2 ) {
34         printf( "%d is less than or equal to %d\n", num1, num2 );
35      } // end if
36
37      if ( num1 >= num2 ) {
38         printf( "%d is greater than or equal to %d\n", num1, num2 );
39      } // end if
40   } // 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 Decision Making: Equality and Relational Operators

- Some of the words we've used in the C programs in this chapter—in particular **`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	

Common Programming Errors

- syntax error
  - a violation of the C grammar rules
  - detected during program translation (compilation)
- run-time error
  - an attempt to perform an invalid operation
  - detected during program execution
- logic errors
  - an error caused by following an incorrect algorithm

Figure 2.16

Compiler Listing of a Program with Syntax Errors

```
221 /* Converts distances from miles to kilometers. */
222
223 #include <stdio.h>          /* printf, scanf definitions  */
266 #define KMS_PER_MILE 1.609  /* conversion constant        */
267
268 int
269 main(void)
270 {
271       double kms
272
273       /* Get the distance in miles. */
274       printf("Enter the distance in miles> ");
***** Semicolon added at the end of the previous source line
275       scanf("%lf", &miles);
***** Identifier "miles" is not declared within this scope
***** Invalid operand of address-of operator
276
277       /* Convert the distance to kilometers. */
278       kms = KMS_PER_MILE * miles;
***** Identifier "miles" is not declared within this scope
279
280       /* Display the distance in kilometers. * /
281       printf("That equals %f kilometers.\n", kms);
282
283       return (0);
284 }
***** Unexpected end-of-file encountered in a comment
***** "}" inserted before end-of-file
```

[2] Problem Solving & Program Design in C, Eighth Edition, R. Hanly & Elliot B. Koffman.

Figure 2.17

A Program with a Run-Time Error

```
111 #include <stdio.h>
262
263 int
264 main(void)
265 {
266         int    first, second;
267         double temp, ans;
268
269         printf("Enter two integers> ");
270         scanf("%d%d", &first, &second);
271         temp = second / first;
272         ans = first / temp;
273         printf("The result is %.3f\n", ans);
274
275         return (0);
276 }

Enter two integers> 14 3
Arithmetic fault, divide by zero at line 272 of routine main
```

[2] Problem Solving & Program Design in C, Eighth Edition, R. Hanly & Elliot B. Koffman.

Figure 2.19

A Program That Produces Incorrect Results Due to & Omission

```
1.  #include <stdio.h>
2.
3.  int
4.  main(void)
5.  {
6.      int    first, second, sum;
7.
8.      printf("Enter two integers> ");
9.      scanf("%d%d", first, second); /* ERROR!! should be &first, &second */
10.     sum = first + second;
11.     printf("%d + %d = %d\n", first, second, sum);
12.
13.     return (0);
14. }

Enter two integers> 14    3
5971289 + 5971297 = 11942586
```

[2] Problem Solving & Program Design in C, Eighth Edition, R. Hanly & Elliot B. Koffman.

Wrap Up

- Every C program has preprocessor directives and a main function.
- The main function contains variable declarations and executable statements.
- C’s data types enable the compiler to determine how to store a value in memory and what operations can be performed on that value.