02_02_Introduction to C Programming

Objectives

In this chapter, you'll:

- Write simple C programs.
- Use simple input and output statements.
- Use the fundamental data types.
- Learn computer memory concepts.
- Use arithmetic operators.
- Learn the precedence of arithmetic operators.
- Write simple decision-making statements.
- Begin focusing on secure C programming practices.

```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.

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 .

```
int main()

void main()

int main(void)

int main(int argc, char* argv[])
```

`int main()` means you will end your program with `return 0;`(or other value,0 is standard for "everything's fine").

`void main()` will return nothing.

`int main(void)` means there is no arguments.

`int main(int argc,char *argv[])` means there are two parameter, one is `int` type and another one is char type pointer array. The first parameter stores the number of command line arguments entered and the second parameter is used to store the arguments.

```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.

```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`.

```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.)

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.

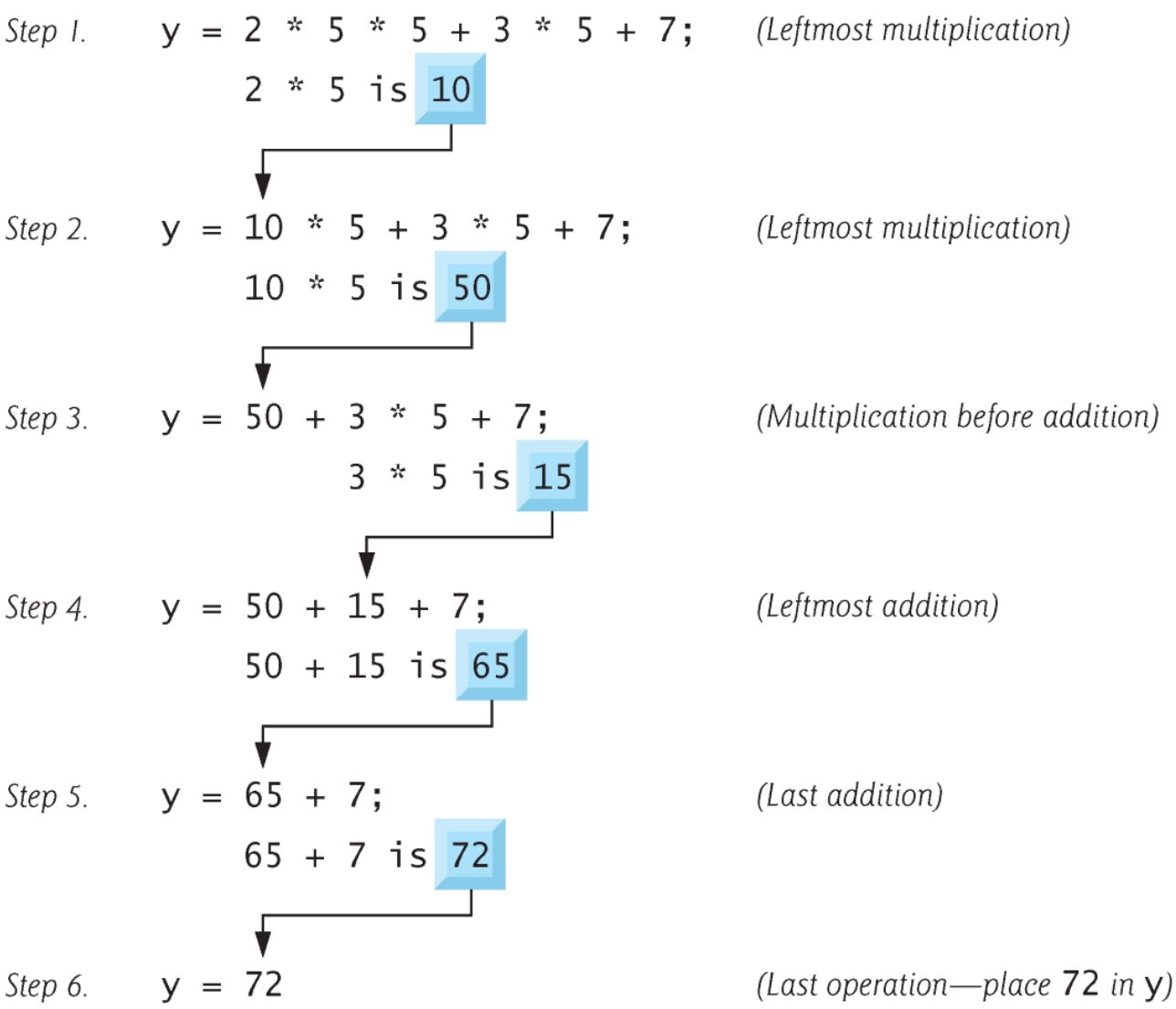

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

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.)

```
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.

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.

Secure C Programming

- CERT C Secure Coding Standard
  - Guidelines that help you avoid programming practices that open systems to attacks

- Avoid Single-Argument `printf`s
  - If you need to display a string that terminates with a newline, use the `puts` function, which displays its string argument followed by a newline character
  - For example

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

  - should be written as:

```
puts( "Welcome to C!" );
```

  - We did not include `\n` in the preceding string because puts adds it automatically.

Secure C Programming (cont.)

- If you need to display a string without a terminating newline character, use `printf` with two arguments.
  - For example

    ```
    printf( "Welcome " );
    ```
  - should be written as:

    ```
    printf( "%s", "Welcome " );
    ```
  - These changes are responsible coding practices that eliminate certain security C vulnerabilities as we get deeper into C