BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-1

Introduction to C Programming

Instructor : PhD, Associate Professor Leyla Muradkhanli

Outline

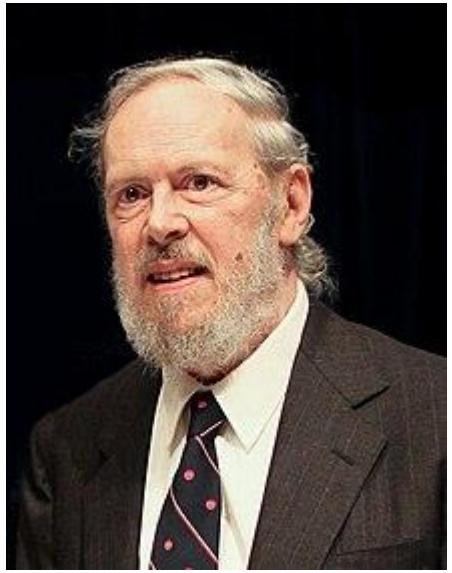
- History of C
- Simple Computer Program in C
- Input and Output Statements
- Arithmetic Operators
- Equality and Relational Operators

Introduction

- The C language facilitates a structured and disciplined approach to computer program design.
- In this topic we introduce C programming and present several examples that illustrate many important features of C.

History of C

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system.
C is the most widely used computer language.



Why to Learn C Programming?

Key advantages of learning C Programming:

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Hello World using C Programming

```c
#include <stdio.h>
int main() {
   /* my first program in C */
   printf("Hello, World! \n");

return 0;
}
```

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" );

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

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

A Simple C Program: Printing a Line of Text

- Lines 1 and 2
  - `/* Fig. 2.1: fig02_01.c`
    `A first program in C */`
- begin with `/*` and end with `*/` indicating that these two lines are a comment.
- You insert comments to document programs and improve program readability.
- Comments do not cause the computer to perform any action when the program is run.

A Simple C Program: Printing a Line of Text

- Comments are ignored by the C compiler and do not cause any machine-language object code to be generated.
- Comments also help other people read and understand your program.

A Simple C Program: Printing a Line of Text

- Some programmers prefer `//` comments because they’re shorter and they eliminate the common programming errors that occur with `/* */` comments.
- Line 3
    - `#include <stdio.h>`
- is a directive to the C preprocessor.

A Simple C Program: Printing a Line of Text

- Lines beginning with # are processed by the preprocessor before the program is compiled.
- Line 3 tells the preprocessor to include the contents of the standard input/output header (`<stdio.h>`) in the program.
- This header contains information used by the compiler when compiling calls to standard input/output library functions such as `printf`.
- Line 6
  - `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.

A Simple C Program: Printing a Line of Text

- C programs contain one or more functions, one of which must be `main`.
- Every program in C begins executing at the function `main`.
- The keyword `int` to the left of `main` indicates that `main` “returns” an integer (whole number) value.

A Simple C Program: Printing a Line of Text

- For now, simply include the keyword `int` to the left of `main` in each of your programs.
- Functions also can receive information when they're called upon to execute.
- The `void` in parentheses here means that `main` does not receive any information.

A Simple C Program: Printing a Line of Text

- A left brace, `{`, begins the body of every function (line 7).
- A corresponding right brace ends each function (line 11).
- This pair of braces and the portion of the program between the braces is called a block.
- 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.
- A string is sometimes called a character string, a message or a literal.

A Simple C Program: Printing a Line of Text

- The entire line, including `printf`, its argument within the parentheses and the semicolon `(;)`, is called a statement.
- Every statement must end with a semicolon (also known as the statement terminator).
- When the preceding `printf` statement is executed, it prints the message `Welcome to C!` on the screen.
- The characters normally print exactly as they appear between the double quotes in the `printf` statement.
- Notice that the characters `\n` were not printed on the screen.
- The backslash (`\`) is called an escape character.
- It indicates that `printf` is supposed to do something out of the ordinary.

A Simple C Program: Printing a Line of Text

- When encountering a backslash in a string, the compiler looks ahead at the next character and combines it with the backslash to form an escape sequence.
- The escape sequence `\n` means newline.
- When a newline appears in the string output by a `printf`, the newline causes the cursor to position to the beginning of the next line on the screen.
- Some common escape sequences are listed in Fig. 2.2.

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. Sound the system bell.
`\\`	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 .

A Simple C Program: Printing a Line of Text

- Because the backslash has special meaning in a string, i.e., the compiler recognizes it as an escape character, we use a double backslash (`\\`) to place a single backslash in a string.
- Printing a double quote also presents a problem because double quotes mark the boundary of a string—such quotes are not printed.
- By using the escape sequence `\"` in a string to be output by `printf`, we indicate that `printf` should display a double quote.

A Simple C Program: Printing a Line of Text

- Line 10
  - `return 0; /* indicate that program ended successfully */`
- is included at the end of every `main` function.
- The keyword `return` is one of several means we'll use to exit a function.
- When the return statement is used at the end of `main` as shown here, the value `0` indicates that the program has terminated successfully.
- The right brace, `}`, (line 12) indicates that the end of `main` has been reached.

A Simple C Program: Printing a Line of Text

- We said that `printf` causes the computer to perform an action.
- As any program executes, it performs a variety of actions and makes decisions.

A Simple C Program: Printing a Line of Text

- Standard library functions like `printf` and `scanf` are not part of the C programming language.
- For example, the compiler cannot find a spelling error in `printf` or `scanf`.
- When the compiler compiles a `printf` statement, it merely provides space in the object program for a “call” to the library function.
- But the compiler does not know where the library functions are—the linker does.
- When the linker runs, it locates the library functions and inserts the proper calls to these library functions in the object program.

A Simple C Program: Printing a Line of Text

- Now the object program is complete and ready to be executed.
- For this reason, the linked program is called an executable.
- If the function name is misspelled, it’s the linker that will spot the error, because it will not be able to match the name in the C program with the name of any known function in the libraries.

A Simple C Program: Printing a Line of Text

- The `printf` function can print `Welcome to C!` several different ways.
- For example, the program of Fig. 2.3 produces the same output as the program of Fig. 2.1.
- This works because each `printf` resumes printing where the previous `printf` stopped printing.
- The first `printf` (line 8) prints `Welcome` followed by a space and the second `printf` (line 9) begins printing on the same line immediately following the space.

```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" );

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

```
Welcome to C!
```

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

A Simple C Program: Printing a Line of Text

- One `printf` can print several lines by using additional newline characters as in Fig. 2.4.
- Each time the `\n` (newline) escape sequence is encountered, output continues at the beginning of the next line.

```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" );

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

```
Welcome
to
C!
```

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

Another Simple C Program: Adding Two Integers

- Our next program (fig. 3.8) 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`.
- [In the input/output dialog of Fig. 2.8, we emphasize the numbers input by the user in **bold.]**

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

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

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

   sum = integer1 + integer2; /* assign total to sum */

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

   return 0; /* indicate that program ended successfully */
} /* 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.)

Another Simple C Program: Adding Two Integers

- 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.
- A variable is a location in memory where a value can be stored for use by a program.
- These definitions specify that the variables `integer1`, `integer2` and `sum` are of type `int`, which means that these variables will hold integer values, i.e., whole numbers such as 7, –11, 0, 31914 and the like.

Another Simple C Program: Adding Two Integers

- All variables must be defined with a name and a data type immediately after the left brace that begins the body of `main` before they can be used in a program.
- The preceding definitions could have been combined into a single definition statement as follows:
    - `int integer1, integer2, sum;`
- but that would have made it difficult to describe the variables in corresponding comments as we did in lines 8–10.
- A variable name in C is any valid identifier.
- An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.

Another Simple C Program: Adding Two Integers

- An identifier can be of any length, but only the first 31 characters are required to be recognized by C compilers according to the C standard.
- C is case sensitive—uppercase and lowercase letters are different in C, so `a1` and `A1` are different identifiers.

Another Simple C Program: Adding Two Integers

- Definitions must be placed after the left brace of a function and before *any executable statements*.
- A syntax error is caused when the compiler cannot recognize a statement.
- The compiler normally issues an error message to help you locate and fix the incorrect statement.
- Syntax errors are violations of the language.
- Syntax errors are also called compile errors, or compile-time errors.

Another Simple C Program: Adding Two Integers

- Line 12
  - `printf( "Enter first integer\n" ); /* prompt */`
- prints the literal `Enter first integer` on the screen and positions the cursor to the beginning of the next line.
- This message is called a prompt because it tells the user to take a specific action.
- 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.

Another Simple C Program: Adding Two Integers

- This `scanf` has two arguments, `"%d"` and `&integer1`.
- The first argument, the format control string, indicates the type of data that should be input by the user.
- The `%d` conversion specifier indicates that the data should be an integer (the letter `d` stands for “decimal integer”).
- The `%` in this context is treated by `scanf` (and `printf` as we’ll see) as a special character that begins a conversion specifier.
- The second argument of `scanf` begins with an ampersand (&)—called the address operator in C—followed by the variable name.

Another Simple C Program: Adding Two Integers

- The ampersand, when combined with the variable name, tells `scanf` the location (or address) in memory at which the variable `integer1` is stored.
- The computer then stores the value for `integer1` at that location.
- The use of ampersand (&) is often confusing to novice programmers or to people who have programmed in other languages that do not require this notation.
- For now, just remember to precede each variable in every call to `scanf` with an ampersand.

Another Simple C Program: Adding Two Integers

- When the computer executes the preceding `scanf`, it waits for the user to enter a value for variable `integer1`.
- The user responds by typing an integer, then pressing the Enter key to send the number to the computer.
- The computer then assigns this number, or value, to the variable `integer1`.
- Any subsequent references to `integer1` in this program will use this same value.
- Functions `printf` and `scanf` facilitate interaction between the user and the computer.
- Because this interaction resembles a dialogue, it is often called conversational computing or interactive computing.

Another Simple C Program: Adding Two Integers

- Line 15
  - `printf( "Enter second integer\n" ); /* prompt */`
- displays the message `Enter second integer` on the screen, then positions the cursor to the beginning of the next line.
- This `printf` also prompts the user to take action.
- The statement
  - `scanf( "%d", &integer2 ); /* read an integer */`
- obtains a value for variable `integer2` from the user.

Another Simple C Program: Adding Two Integers

- The assignment statement in line 18
  - `sum = integer1 + integer2; /* assign total to sum */`
- calculates the sum of variables `integer1` and `integer2` and assigns the result to variable `sum` using the assignment operator =.
- The statement is read as, "`sum` *gets the value of* `integer1 + integer2`*." Most calculations are performed in assignments.*
- The = operator and the + operator are called binary operators because each has two operands.
- The + operator's two operands are `integer1` and `integer2`.
- The = operator's two operands are `sum` and the value of the expression `integer1 + integer2`.

Another Simple C Program: Adding Two Integers

- 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.
- This `printf` has two arguments, `"Sum is %d\n"` and `sum`.
- The first argument is the format control string.
- It contains some literal characters to be displayed, and it contains the conversion specifier `%d` indicating that an integer will be printed.
- The second argument specifies the value to be printed.
- Notice that the conversion specifier for an integer is the same in both `printf` and `scanf`.

Another Simple C Program: Adding Two Integers

- We could have combined the previous two statements into the statement
  - `printf( "Sum is %d\n", integer1 + integer2 );`
- Line 22
  - `return 0; /* indicate that program ended successfully */`
- passes the value 0 back to the operating-system environment in which the program is being executed.
- This value indicates to the operating system that the program executed successfully.
- For information on how to report a program failure, see the manuals for your particular operating-system environment.
- The right brace, `}`, at line 24 indicates that the end of function `main` has been reached.

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.
- In the addition program of Fig. 2.5, when the statement (line 13)
    - `scanf( "%d", &integer1 ); /* read an integer */`
- is executed, the value typed 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.

Memory Concepts

- Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, placing a new value into a memory location is said to be destructive.
- Returning to our addition program again, when the statement (line 16)
    - `scanf( "%d", &integer2 ); /* read an integer */`
- executes, suppose the user enters the value `72`.
- This value is placed into location `integer2`, and memory appears as in Fig. 2.7.
- These locations are not necessarily adjacent in memory.

Memory Concepts

- Once the program has obtained values for `integer1` and `integer2`, it adds these values and places the sum into variable `sum`.
- The statement (line 18)
    - `sum = integer1 + integer2; /* assign total to sum */`
- that performs the addition also replaces whatever value was stored in `sum`.

Memory Concepts

- This occurs when the calculated sum of `integer1` and `integer2` is placed into location `sum` (destroying the value already in `sum`).
- After `sum` is calculated, memory appears as in Fig. 2.8.
- The values of `integer1` and `integer2` appear exactly as they did before they were used in the calculation.

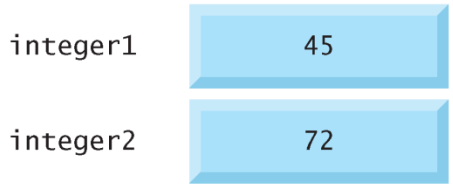


Fig. 2.7 | Memory locations after both variables are input.

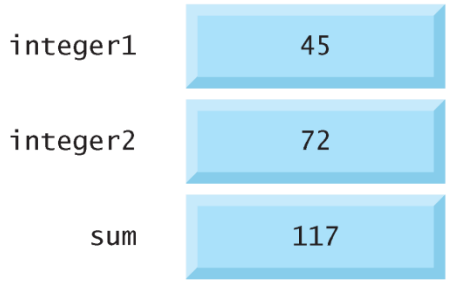


Fig. 2.8 | Memory locations after a calculation.

Memory Concepts

- They were used, but not destroyed, as the computer performed the calculation.
- Thus, when a value is read from a memory location, the process is said to be nondestructive.

Arithmetic in C

- The C arithmetic operators are summarized in Fig. 2.9.
- Note the use of various special symbols not used in algebra.
- The asterisk (`*`) indicates multiplication and the percent sign (`%`) denotes the remainder operator, which is introduced below.
- In algebra, if we want to multiply *a times b, we can simply place these single-letter variable names side by side as in ab.*
- In C, however, if we were to do this, `ab` would be interpreted as a single, two-letter name (or identifier).
- Therefore, C requires that multiplication be explicitly denoted by using the * operator as in `a * b`.

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.

Arithmetic in C

- The arithmetic operators are all binary operators.
- For example, the expression `3 + 7` contains the binary operator `+` and the operands `3` and `7`.
- Integer division yields an integer result.
- For example, the expression `7 / 4` evaluates to `1` and the expression `17 / 5` evaluates to `3`.
- C provides the remainder operator, `%`, which yields the remainder after integer division.
- The remainder operator is an integer operator that can be used only with integer operands.
- The expression `x % y` yields the remainder after `x` is divided by `y`.
- Thus, `7 % 4` yields `3` and `17 % 5` yields `2`.

Arithmetic in C

- Arithmetic expressions in C must be written in straight-line form to facilitate entering programs into the computer.
- Thus, expressions such as "`a` divided by `b`" must be written as `a/b` so that all operators and operands appear in a straight line.
- The algebraic notation

- is generally not acceptable to compilers, although some special-purpose software packages do support more natural notation for complex mathematical expressions.

Arithmetic in C

- Parentheses are used in C expressions in the same manner as in algebraic expressions.
- For example, to multiply `a` times the quantity `b + c` we write `a * ( b + c )`.

Arithmetic in C

- 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 in algebra:
  - Operators in expressions contained within pairs of parentheses are evaluated first. Thus, *parentheses may be used to force the order of evaluation to occur in any sequence you desire. Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as*
    - `( ( a + b ) + c )`
  - the operators in the innermost pair of parentheses are applied first.

Arithmetic in C

- Multiplication, division and remainder operations are applied first. If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right. Multiplication, division and remainder are said to be on the same level of precedence.
- Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.

Arithmetic in C

- The rules of operator precedence specify the order C uses to evaluate expressions. When we say evaluation proceeds from left to right, we're referring to the associativity of the operators.
- We'll see that some operators associate from right to left.
- Figure 2.10 summarizes these rules of 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	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Fig. 2.10 | Precedence of arithmetic operators.

Arithmetic in C

- Figure 2.11 illustrates the order in which the operators are applied.

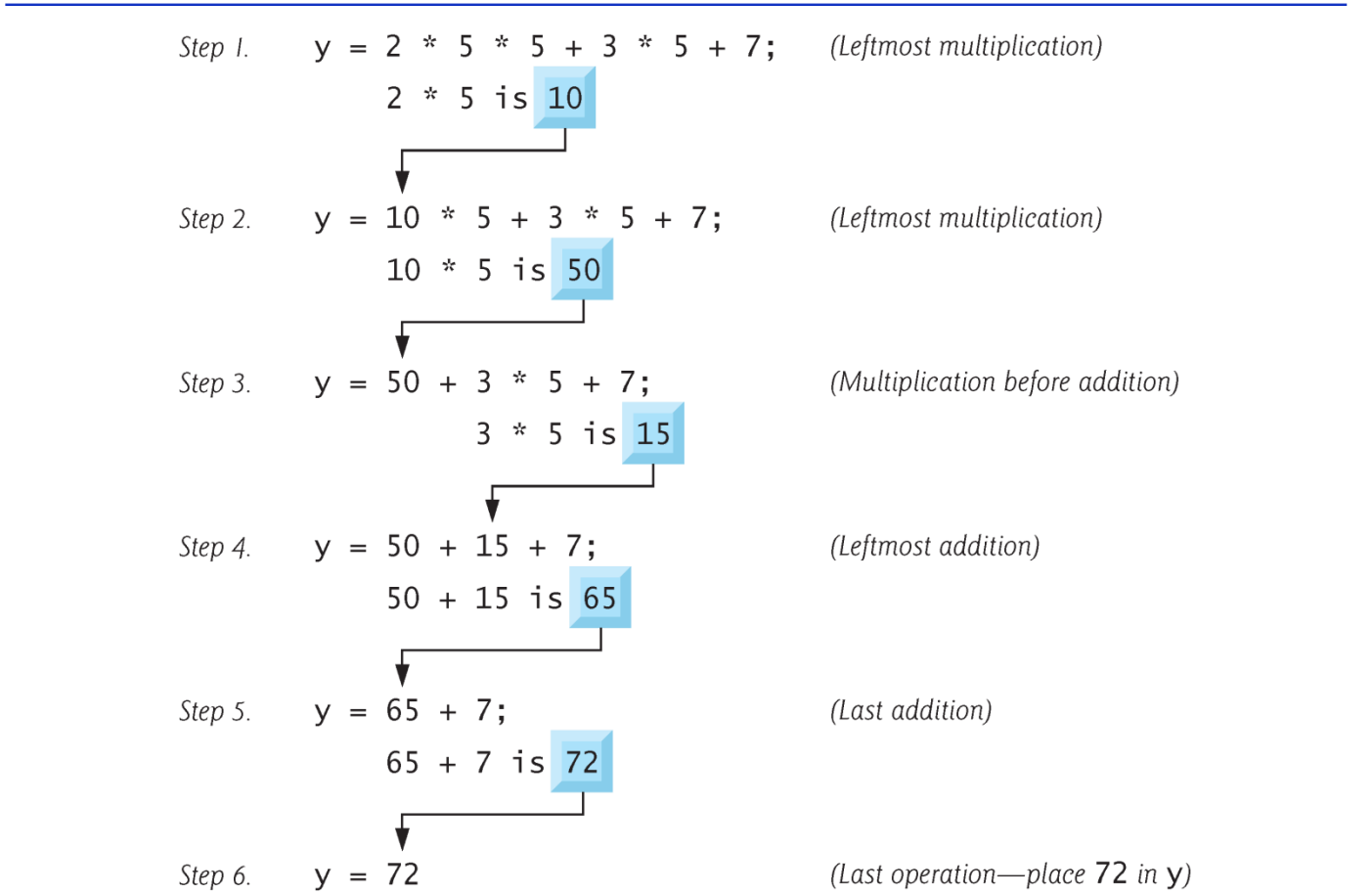


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

Arithmetic in C

- As in algebra, it is acceptable to place unnecessary parentheses in an expression to make the expression clearer.
- These are called redundant parentheses.

Decision Making: Equality and Relational Operators

- Executable C statements either perform actions (such as calculations or input or output of data) or make decisions (we'll soon see several examples of these).
- We might make a decision in a program, for example, to determine if a person's grade on an exam is greater than or equal to 60 and if it is to print the message "Congratulations! You passed."
- This section introduces a simple version of 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.

Equality and Relational Operators

- If the condition is met (i.e., the condition is true) the statement in the body of the `if` statement is executed.
- If the condition is not met (i.e., the condition is false) the body statement is not executed.
- Whether the body statement is executed or not, after the `if` statement completes, execution proceeds with the next statement after the `if` statement.
- Conditions in `if` statements are formed by using the equality operators and relational operators summarized in Fig. 2.12.

Equality and Relational Operators

- The relational operators all have the same level of precedence and they associate left to right.
- The equality operators have a lower level of precedence than the relational operators and they also associate left to right.
- In C, a condition may actually be any expression that generates a zero (false) or nonzero (true) value.

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

Fig. 2.12 | Equality and relational operators.

Equality and Relational Operators

- To avoid this confusion, the equality operator should be read “double equals” and the assignment operator should be read “gets” or “is assigned the value of.”
- As we’ll soon see, confusing these operators may not necessarily cause an easy-to-recognize compilation error, but may cause extremely subtle logic errors.

Equality and Relational Operators

- Figure 2.13 uses six `if` statements to compare two numbers input 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     int num1; /* first number to be read from user  */
10    int num2; /* second number to be read from user */
11
12    printf( "Enter two integers, and I will tell you\n" );
13    printf( "the relationships they satisfy: " );
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
21    if ( num1 != num2 ) {
22       printf( "%d is not equal to %d\n", num1, num2 );
23    } /* end if */
```

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

```
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
41      return 0; /* indicate that program ended successfully */
42   } /* 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: 12 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.)

Equality and Relational Operators

- The program uses `scanf` (line 15) to input two numbers.
- Each conversion specifier has a corresponding argument in which a value will be stored.
- The first `%d` converts a value to be stored in variable `num1`, and the second `%d` converts a value to be stored in variable `num2`.
- Indenting the body of each `if` statement and placing blank lines above and below each `if` statement enhances program readability.

Equality and Relational Operators

- A left brace, `{`, begins the body of each `if` statement (e.g., line 17).
- A corresponding right brace, `}`, ends each `if` statement's body (e.g., line 19).
- Any number of statements can be placed in the body of an `if` statement.
- The comment (lines 1–3) in Fig. 2.13 is split over three lines.
- In C programs, white space characters such as tabs, newlines and spaces are normally ignored.
- So, statements and comments may be split over several lines.
- It is not correct, however, to split identifiers.

Equality and Relational Operators

- Figure 2.14 lists the precedence of the operators introduced in this chapter.
- Operators are shown top to bottom in decreasing order of precedence.
- The equals sign is also an operator.
- All these operators, with the exception of the assignment operator =, associate from left to right.
- The assignment operator (=) associates from right to left.

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.

Equality and Relational Operators

- Some of the words we have used in the C programs in this chapter—in particular `int`, `return` and `if`—are keywords or reserved words of the language.
- Figure 2.15 contains the C keywords.
- 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`	`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`

Fig. 2.15 | C's keywords.