Chapter 2 Introduction to C Programming

C How to Program

Introduction

- The C language facilitates a structured and disciplined approach to computer program design.

- In this chapter we introduce C programming and present several examples that illustrate many important features of C.

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).

```
1  // Fig. 2.1: fig02_01.c
2  // A first program in C.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9  } // end function main
```

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.
- 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 (Cont.)

- 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.
- You can also use `/*...*/` multi-line comments in which everything from `/* on the first line to */` at the end of the line is a comment.

#include – Preprocessor directive

- Line 3
  - `#include <stdio.h>`
- is a directive to the C preprocessor.
- Lines beginning with # are processed by the preprocessor before compilation.
- 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.

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

Blank Lines and White Space

- Line 4 is a blank line. You use blank lines, space characters and tab characters to make programs easier to read.
- Together, these characters are known as white space. White-space characters are ignored by the compiler.

The `main` Function

- 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 (Cont.)

- 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.
- The void in parentheses here means that main does not receive any information.
- Functions will be explained in Chapter 5.

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

- 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.

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.
- A string is sometimes called a character string, a message or a literal.

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

- The entire line, including the printf function (the "f" stands for "formatted"), its argument within the parentheses and the semicolon (;), is called a statement.
- Every statement must end with a semicolon (;).
- When the preceding printf statement is executed, it prints the message `Welcome to C!` on the screen.

Escape Sequences

- 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 (Cont.)

- 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. 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 .

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

- 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 boundaries 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.
- The right brace, }, (line 9) indicates that the end of `main` has been reached.

The Linker and Executables

- printf and scanf are Standard library functions.
- 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.

The Linker and Executables (cont.)

- 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 (Cont.)

- Each time the \n (newline) escape sequence is encountered, output continues at the beginning of the next line.

```
1  // Fig. 2.4: fig02_04.c
2  // Printing multiple lines with a single printf.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     printf( "Welcome\nto\nC!\n" );
9  } // end function main
```

```
Welcome
to
C!
```

```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
   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
} // end function main
```

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

Adding Two Integers

- This program 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.

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

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

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 */
    ```

- The names `integer1`, `integer2` and `sum` are the names of variables—locations in memory where values 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 they'll hold integer values, i.e., whole numbers such as 7, –11, 0, 31914 and the like.
    ```

Adding Two Integers (Cont.)

- All variables must be defined with a name and a data type 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 with corresponding comments as we did in lines 8–10.

Adding Two Integers (Cont.)

Identifiers and Case Sensitivity

- 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.
- C is case sensitive—uppercase and lowercase letters are different in C, so `a1` and `A1` are different identifiers.

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 because it tells the user to take a specific action.

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.

Adding Two Integers (Cont.)

- This scanf has two arguments, "%d" and &integer1.
- "%d", is a format control string, it 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 a special character that begins a conversion specifier.

Adding Two Integers (Cont.)

- The second argument of scanf begins with an ampersand (&)—called the address operator in C—followed by the variable name.
- The &, 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 that the user enters for integer1 at that location.

Adding Two Integers (Cont.)

- 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.

Adding Two Integers (Cont.)

- 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.

- Line 16
  - `scanf( "%d", &integer2 ); /* read an integer */`

  obtains a value for variable `integer2` from the user.

Adding Two Integers (Cont.)

Assignment Statement

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

  calculates the total of variables integer1 and integer2 and assigns the result to variable sum using the "=" operator.

- The statement is read as, "`sum` *gets* the value of `integer1 + integer2`."

- The "=" operator and the "+" operator are called binary operators because each has two operands.

Adding Two Integers (Cont.)

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.

- 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.

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.

Type in CodeBlocks

```
1  // Fig. 2.5: fig02_05.c
2  // Addition program
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     int integer1; // first number to be entered by user
9     int integer2; // second number to be entered by user
10    int sum; // variable in which sum will be stored
11
12    printf( "Enter first integer\n" ); // prompt
13    scanf( "%d", &integer1 ); // read an integer
14
15    printf( "Enter second integer\n" ); // prompt
16    scanf( "%d", &integer2 ); // read an integer
17
18    sum = integer1 + integer2; // assign total to sum
19
20    printf( "Sum is %d\n", sum ); // print sum
21
22    return 0;
23 } // end function main
24
```

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.
- 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`

Memory Concepts (Cont.)

- Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, this process is said to be destructive.

- When the statement
  - `scanf( "%d", &integer2 ); /* read an integer */`

  executes, suppose the user enters the value `72`.

- This value is placed into location `integer2`

- These locations are not necessarily adjacent in memory.

Memory Concepts (Cont.)

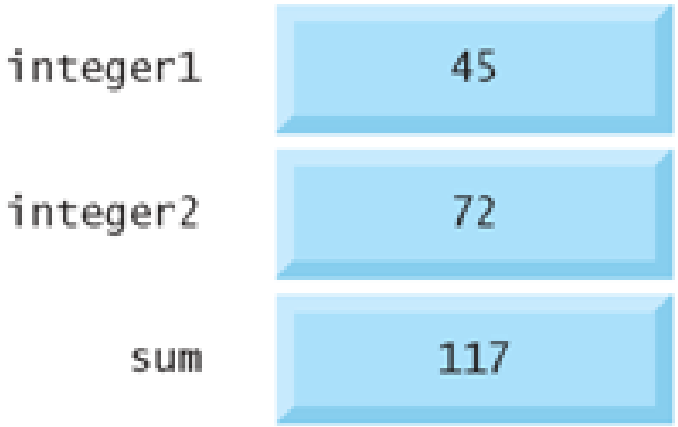
- Once the program has obtained values for `integer1` and `integer2`, it adds these values and places the total into variable `sum`.

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

- that performs the addition also replaces whatever value was stored in `sum`.

Memory Concepts (Cont.)

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



Memory Concepts (Cont.)

- 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

- Most C programs perform calculations using the C arithmetic operators (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.
- To multiply a times b, C requires that multiplication be explicitly denoted by using the * operator as in a * b.
- The arithmetic operators are all binary operators.
- For example, the expression 3 + 7 contains the binary operator + and the operands 3 and 7.

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 (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`.
- 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`.

Common Programming Error 2.7

An attempt to divide by zero is normally undefined on computer systems and generally results in a fatal error, i.e., an error that causes the program to terminate immediately without having successfully performed its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.

Arithmetic Expressions in Straight-Line Form

- 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

  $\frac{a}{b}$

  is generally not acceptable to compilers.

Parentheses for Grouping Subexpressions

- 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 )`.

Rules of Operator Precedence

- 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. 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.

Precedence of arithmetic operators

- The rules of operator precedence specify the order C uses to evaluate expressions.

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.

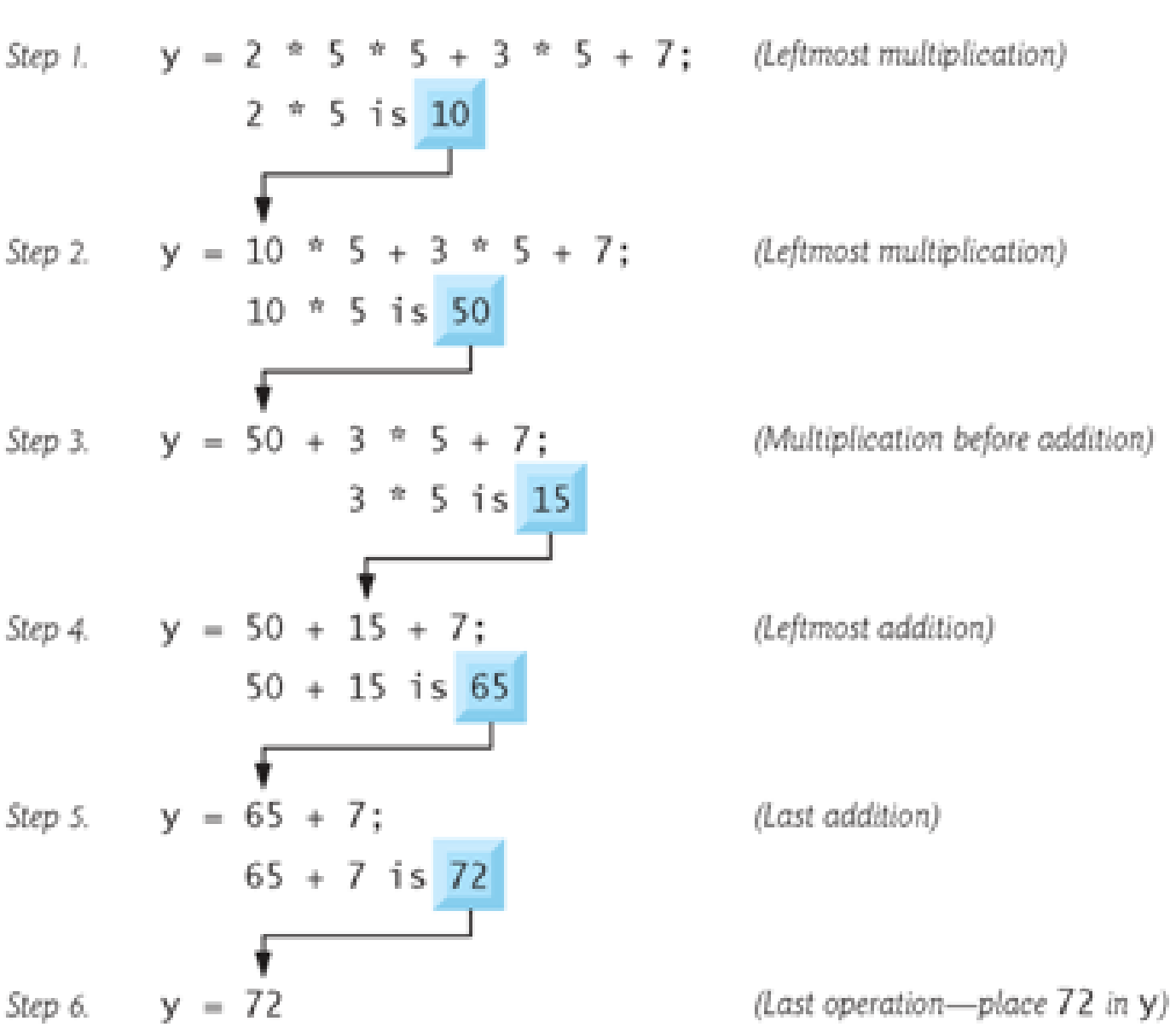


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

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 might make a decision in a program, for example, to determine whether a person's grade on an exam is greater than or equal to 60 and whether the program should 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.

Decision Making: Equality and Relational Operators

- If the condition is true (i.e., the condition is met) the statement in the body of the `if` statement is executed.
- If the condition is false (i.e., the condition isn't met) 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.

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.

Decision Making: 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*.

Decision Making: Equality and Relational Operators

Comparing Numbers

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

The if statement compares the values of variables `num1` and `num2` to test for equality.

- If the values are equal, the statement displays a line of text indicating that the numbers are equal.

Decision Making: Equality and Relational Operators

- A left brace, `{`, begins the body of each `if` statement
- A corresponding right brace, `}`, ends each `if` statement's body
- Any number of statements can be placed in the body of an `if` statement.

Decision Making: Equality and Relational Operators

- All these operators, with the exception of the assignment operator =, associate from left to right.
- The assignment operator (=) associates from right to left.
- If you are uncertain about the order of evaluation in a complex expression, use parentheses to group expressions or break the statement into several simpler statements.

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.
- 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 standard

```
_Bool  _Complex  _Imaginary  inline  restrict
```

Keywords added in C11 draft standard

```
_Alignas  _Alignof  _Atomic  _Generic  _Noreturn  _Static_assert  _Thread_local
```

Fig. 2.15 | C's keywords.

Type in CodeBlocks

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

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
```

Self-Review exercise

- Write a program that reads in two integers and determines and prints whether the first is a multiple of the second.

- Define variables x and y of type int.
- Read the values using scanf
- Use the if statement to determine whether the x is a multiple of y.