Chapter 2

Introduction to C Programming
Objectives of This Chapter

- To write simple computer programs in C programming Language
  *printf* statement and *escape sequences* for printing will be presented.
- To use simple input and output statements.
  *Scanf* statement for inputting value will be presented.
- To use the fundamental data types.
  *Integer, character, string and float* type of variables will be studied.
- To use arithmetic operators.
  +, -, *, /, and % operators will be learned.
- The precedence of arithmetic operators.
  *Using ( ) in the arithmetic expressions and priorities.*
- To write simple decision-making statements.
  *Simple if* statement will be presented.
2.2 A Simple C Program: Printing a Line of Text

Simplest program one may consider is the printing text to the screen.

```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 */
```

Screen/Terminal Output of the Program

```
Welcome to C!
```

C programs are case sensitive, instructions/functions must be write always in lowercases.
Even though this program is a simple program, it illustrates several important features of the C Programming language.
2.2 A Simple C Program: Printing a Line of Text

/* Fig. 2.1: fig02_01.c */

A first program in C */

Lines between */ and */ are comment lines. Not part of the program, but it help us to add some notes to program. So that the program will be more readable. Also // can be used for commenting purposes but only used for one line.

#include <stdio.h>

Lines beginning with # are called “preprocessor directives” which are processed by the compiler before starting compiler. #include<stdio.h> tells the preprocessor to include the standard input/output Library/header (<stdio.h>) in this program.

int main( void )
{

This is the beginning of the program. Compiler start compiling the instructions from this point. Main function is always followed by { “left - brace”.

printf( "Welcome to C!\n" );
The printf function is used for printing to standard-output-device usually screen. Notice that all instructions in C ends with “;”. It prints the stuffs inside the Quotation marks.

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

} /* end function main */

} “right-brace” is the end of the main function. This is where the program finish/terminates. return 0; is used to indicate the program has finished successfully.
Screen output of the printing function `printf` function is as below

```
Welcome to C!
```

As you may notice that the “\n” is not printed to the screen. The expressions inside `printf` starting with “\” (Back Slashes) are called **Escape Sequences**.

**Escape Sequences** are not printed screen but only used to control way of the printing.

Here “\n” is the newline escape sequence indicating that Cursor will go to the next line at this point.

There are other escape sequences available to do the printing in more flexible way.
# Commonly Used Escape Sequences in C

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Newline. Position the cursor at the beginning of the next line.</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab. Move the cursor to the next tab stop.</td>
</tr>
<tr>
<td>\a</td>
<td>Alert. Sound the system bell.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Backslash. Insert a backslash character in a string.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote. Insert a double-quote character in a string.</td>
</tr>
</tbody>
</table>

**Fig. 2.2** Some common escape sequences.

**Escape Sequences are used to control way of the printing.**
# Important Notes on the Choice of Variable Names

## List of Reserved Words in C

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>const</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>do</td>
</tr>
</tbody>
</table>

*Keywords added in C99*

_Bool  _Complex  _Imaginary  inline  restrict*

---

**Fig. 2.15**  |  C’s keywords.

*These words can’t be declared as variable names*

**Example:**  int break;  /*declaration is not allowed.*/
### 2.5 Arithmetic in C

#### Common Arithmetic Operations in C

<table>
<thead>
<tr>
<th>C operation</th>
<th>Arithmetic operator</th>
<th>Algebraic expression</th>
<th>C expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>$f + 7$</td>
<td>f + 7</td>
</tr>
<tr>
<td>Subtraction</td>
<td>−</td>
<td>$p - c$</td>
<td>p - c</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>$bm$</td>
<td>b * m</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>$x/y$ or $\frac{x}{y}$ or $x + y$</td>
<td>x / y</td>
</tr>
<tr>
<td>Remainder</td>
<td>%</td>
<td>$r \mod s$</td>
<td>r % s</td>
</tr>
</tbody>
</table>

**Fig. 2.9**  | Arithmetic operators.

The C programming language understand this syntax. You can use any of these operation in given format.
2.6 Decision Making: Equality and Relational Operators

In many programs, we need to make a decision, for example, to determine if a person’s grade on an exam is greater than or equal to 50 and if it is to print the message “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. In the next chapter, we will introduce more complex nested if statement.

```c
if (condition) {
    printf(“PASSED”);
}
```
### 2.6 Decision Making: Equality and Relational Operators

#### List operators used in Conditional Statement

<table>
<thead>
<tr>
<th>Algebraic equality or relational operator</th>
<th>C equality or relational operator</th>
<th>Example of C condition</th>
<th>Meaning of C condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Equality operators</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>≠</td>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
<tr>
<td><strong>Relational operators</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>≥</td>
<td>&gt;=</td>
<td>x &gt;= y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>≤</td>
<td>&lt;=</td>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
</tbody>
</table>

**Fig. 2.12** | Equality and relational operators.

These conditions are used mostly in “if statements”.

There are many other conditional statements in which also these operators can be used (Example: while, for – Repetitions - etc.)

We will learn these statements in the coming chapters.
```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 )
{
    int num1; /* first number to be read from user */
    int num2; /* second number to be read from user */

    printf( "Enter two integers, and I will tell you\n" );
    printf( "the relationships they satisfy: " );

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

    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 */

    return 0; /* indicate that program ended successfully */
} /* end function main */
```
2.3 Write a single C statement to accomplish each of the following:

a) Define the variables c, thisVariable, q76354 and number to be of type int.

b) Prompt the user to enter an integer. End your prompting message with a colon (:) followed by a space and leave the cursor positioned after the space.

c) Read an integer from the keyboard and store the value entered in integer variable a.

d) If number is not equal to 7, print "The variable number is not equal to 7."

e) Print the message "This is a C program." on one line.

f) Print the message "This is a C program." on two lines so that the first line ends with C.

g) Print the message "This is a C program." with each word on a separate line.

h) Print the message "This is a C program." with the words separated by tabs.
Self-Review Exercises

2.6 Identify and correct the errors in each of the following statements:

a) `printf( "The value is %d\n", &number );`

b) `scanf( "%d%d", &number1, number2 );`

c) `if ( c < 7 ){
    printf( "C is less than 7\n" );
}

d) `if ( c => 7 ) {
    printf( "C is equal to or less than 7\n" );
}`
Exercises

2.14 Given the equation \( y = ax^3 + 7 \), which of the following, if any, are correct C statements for this equation?

a) \( y = a * x * x * x + 7; \)
b) \( y = a * x * x * ( x + 7 ); \)
c) \( y = ( a * x ) * x * ( x + 7 ); \)
d) \( y = ( a * x ) * x * x + 7; \)
e) \( y = a * ( x * x * x ) + 7; \)
f) \( y = a * x * ( x * x + 7 ); \)
Exercises

2.7 Identify and correct the errors in each of the following statements. (*Note: There may be more than one error per statement.*)

a) scanf("d", value);

b) printf("The product of %d and %d is %d\n", x, y);

c) firstNumber + secondNumber = sumOfNumbers

d) if ( number => largest )
   largest == number;

e) /* Program to determine the largest of three integers */

f) Scanf("%d", anInteger);

g) printf("Remainder of %d divided by %d is\n", x, y, x % y);

h) if ( x = y );
   printf("%d is equal to %d\n", x, y);

i) print("The sum is %d\n", x + y);

j) Printf("The value you entered is: %d\n", &value);
Exercises

2.12 What, if anything, prints when each of the following statements is performed? If nothing prints, then answer "Nothing." Assume \( x = 2 \) and \( y = 3 \).

a) `printf( "%d", x );`

b) `printf( "%d", x + x );`

c) `printf( "%x=" );`

d) `printf( "%x=%d", x );`

e) `printf( "%d = %d", x + y, y + x );`

f) `z = x + y;`

g) `scanf( "%d%d", &x, &y );`

h) `/* printf( "%x + y = %d", x + y ); */`
i) `printf( "%\n" );`
Exercises

2.18  *(Comparing Integers)* Write a program that asks the user to enter two integers, obtains the numbers from the user, then prints the larger number followed by the words “is larger.” If the numbers are equal, print the message “These numbers are equal.” Use only the single-selection form of the if statement you learned in this chapter.
Exercise: Data Types

What would be the best variable type for the quantities given below?

a) Area of a circle in square inches.

b) Number of cars passing through an intersection in an hour.

c) The first letter of your name.

d) Number of Students at school.

e) A letter grade on transcript.

f) The average number of school days a child is absent each year.

g) The class-average of an exam as out of 100.
Exercise: Input and Output Statements

How does the computer decide how many data values to get from input device when a `scanf` operation is performed?

Hint: Think of `scanf` (“%d%d”,&var1,&var2);
Exercises

2.20  (Diameter, Circumference and Area of a Circle) Write a program that reads in the radius of a circle and prints the circle’s diameter, circumference and area. Use the constant value 3.14159 for \( \pi \). Perform each of these calculations inside the printf statement(s) and use the conversion specifier %f. [Note: In this chapter, we have discussed only integer constants and variables. In Chapter 3 we’ll discuss floating-point numbers, i.e., values that can have decimal points.]
Exercises

2.22 What does the following code print?

```c
printf("\n**\n***\n****\n*****\n\n");
```
Exercises

2.24  (Odd or Even) Write a program that reads an integer and determines and prints whether it is odd or even. [Hint: Use the remainder operator. An even number is a multiple of two. Any multiple of two leaves a remainder of zero when divided by 2.]
2.30  *(Separating Digits in an Integer) Write a program that inputs one five-digit number, separates the number into its individual digits and prints the digits separated from one another by three spaces each. *[Hint: Use combinations of integer division and the remainder operation.] For example, if the user types in 42139, the program should print

4  2  1  3  9
Exercises

2.31  (Table of Squares and Cubes) Using only the techniques you learned in this chapter, write a program that calculates the squares and cubes of the numbers from 0 to 10 and uses tabs to print the following table of values:

<table>
<thead>
<tr>
<th>number</th>
<th>square</th>
<th>cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>216</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>343</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>729</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>