

4

C Program Control



*Not everything that can be counted counts, and
not every thing that counts can be counted.*

—Albert Einstein

Who can control his fate?

—William Shakespeare

The used key is always bright.

—Benjamin Franklin



Intelligence... is the faculty of making artificial objects, especially tools to make tools.

—Henri Bergson

Every advantage in the past is judged in the light of the final issue.

—Demosthenes



OBJECTIVES

In this chapter you will learn:

- The essentials of **counter-controlled repetition**.
- To use the **for** and **do...while** repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the **switch selection statement**.
- To use the **break** and **continue** program control statements to alter the flow of control.
- To use the **logical operators** to form complex conditional expressions in control statements.
- To avoid the consequences of confusing the **equality** and **assignment** operators.



- 4.1 Introduction
- 4.2 Repetition Essentials
- 4.3 Counter-Controlled Repetition
- 4.4 `for` Repetition Statement
- 4.5 `for` Statement: Notes and Observations
- 4.6 Examples Using the `for` Statement
- 4.7 `switch` Multiple-Selection Statement
- 4.8 `do...while` Repetition Statement
- 4.9 `break` and `continue` Statements
- 4.10 Logical Operators
- 4.11 Confusing Equality (`==`) and Assignment (`=`) Operators
- 4.12 Structured Programming Summary



• ساختار تکرار که حداقل یک بار اجرا می‌شود: **do...while**

• **switch**: بیانیه انتخاب چندگانه.

• دستور **break**: برای خروج فوری و سریع از برخی ساختارهای کنترلی استفاده می‌شود.

• دستور **continue**: برای نادیده گرفتن بقیه بدنه یک ساختار تکرار و ادامه با تکرار بعدی حلقه استفاده می‌شود.

4.1 Introduction

This chapter introduces

- Additional repetition control structures
 - **for**
 - **do...while**
- **switch** multiple selection statement
- **break** statement
 - Used for exiting immediately and rapidly from certain control structures
- **continue** statement
 - Used for skipping the remainder of the body of a repetition structure and proceeding with the next iteration of the loop



4.2 Repetition Essentials

حلقه

گروهی از دستورات که کامپیوتر به طور مکرر در حالی که یک شرط برقرار است، اجرا می‌کند.

■ Loop

- Group of instructions computer executes repeatedly while some condition remains **true**

تکرار کنترل شده با شمارنده

تکرار معین: تعداد دفعات اجرای حلقه مشخص است.
متغیر کنترلی برای شمارش تکرارها استفاده می‌شود.

■ Counter-controlled repetition

- Definite repetition: know how many times loop will execute
- Control variable used to count repetitions

■ Sentinel-controlled repetition

- Indefinite repetition
- Used when number of repetitions not known
- Sentinel value indicates "end of data"

تکرار کنترل شده با سنتیتل

تکرار نامعین: زمانی استفاده می‌شود که تعداد تکرارها مشخص نیست.
مقدار سنتیتل نشان‌دهنده "پایان داده‌ها" است.



4.3 Counter-Controlled Repetition

- **Counter-controlled repetition requires**

- The name of a control variable (or loop counter)
- The initial value of the control variable
- An increment (or decrement) by which the control variable is modified each time through the loop
- A condition that tests for the final value of the control variable (i.e., whether looping should continue)

تکرار کنترل شده با شمارنده نیازمند:

- نام یک متغیر کنترلی (یا شمارنده حلقه)
- مقدار اولیه متغیر کنترلی
- یک افزایش (یا کاهش) که متغیر کنترلی هر بار در حلقه تغییر می کند
- یک شرط که برای مقدار نهایی متغیر کنترلی آزمایش می کند (یعنی آیا باید حلقه ادامه یابد یا خیر)



4.3 Counter-Controlled Repetition

■ Example:

```
int counter = 1;           // initialization
while ( counter <= 10 ) { // repetition condition
    printf( "%d\n", counter );
    ++counter;           // increment
}
```

— The statement

- int counter = 1;**
- **Names counter**
- **Defines it to be an integer**
- **Reserves space for it in memory**
- **Sets it to an initial value of 1**

دستور `int counter = 1;` متغیر `counter` را تعریف و مقدار اولیه آن را
قرار می‌دهد.

شرط `(counter <= 10)` تا زمانی که مقدار `counter` کمتر یا
مساوی ۱۰ است، حلقه را اجرا می‌کند.

در هر تکرار، `printf("%d\n", counter)` مقدار فعلی `counter` را چاپ
می‌کند.

سپس `++counter` مقدار `counter` را به اندازه ۱ افزایش می‌دهد.
این نوع تکرار به عنوان "تکرار کنترل شده با شمارنده" شناخته
می‌شود، زیرا تعداد دفعات اجرای حلقه از قبل مشخص است.



Outline

fig04_01.c

```
1 /* Fig. 4.1: fig04_01.c
2  Counter-controlled repetition */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int counter = 1; /* initialization */ ←
9
10    while ( counter <= 10 ) { /* repetition condition */
11        printf ( "%d\n", counter ); /* display counter */
12        ++counter; /* increment */
13    } /* end while */
14
15    return 0; /* indicate program ended successfully */
16
17 } /* end function main */
```

Definition and assignment are performed simultaneously

1
2
3
4
5
6
7
8
9
10



4.3 Counter-Controlled Repetition

- Condensed code

- C Programmers would make the program more concise
- Initialize counter to 0

```
- while ( ++counter <= 10 )
    printf( "%d\n", counter );
```

در این کد فشرده شده:

متغیر counter ابتدا به مقدار ۰ مقداردهی می‌شود.

در شرط while، ابتدا ++counter اجرا می‌شود که باعث افزایش مقدار counter به ۱ می‌شود، سپس این مقدار با ۱۰ مقایسه می‌شود.

در هر تکرار، printf("%d\n", counter) مقدار فعلی counter را چاپ می‌کند.
این کد به همان صورت قبلی عمل می‌کند، اما به صورت فشرده‌تر نوشته شده است.



Common Programming Error 4.1

Because **floating-point** values may be approximate, controlling counting loops with floating-point variables may result in imprecise counter values and inaccurate tests for termination.



Error-Prevention Tip 4.1

Control counting loops with **integer values**.

به دلیل اینکه مقادیر اعشاری ممکن است تقریبی باشند، استفاده از متغیرهای اعشاری برای کنترل حلقه‌های شمارشی ممکن است منجر به مقادیر شمارنده نادقيق و آزمون‌های پایان نادقيق شود.

نکات کلیدی:

- مقادیر اعشاری ممکن است حاوی خطاهای گرد کردن کوچک باشند.
- این خطاهای می‌توانند باعث مقادیر شمارنده نادقيق در طول تکرارها شوند.
- همچنین، آزمون‌های پایان حلقه که بر اساس متغیرهای اعشاری هستند، ممکن است قابل اعتماد نباشند.
- حلقه‌ها ممکن است زودتر یا دیرتر از انتظار پایان یابند.
- توصیه می‌شود از متغیرهای صحیح (**integer**) برای کنترل حلقه‌های شمارشی استفاده شود تا دقیق‌تری داشته باشند.



Good Programming Practice 4.1

Indent the statements in the body of each control statement.



Good Programming Practice 4.2

Put a **blank line before and after** each control statement to make it stand out in a program.



Good Programming Practice 4.3

Too many levels of nesting can make a program difficult to understand. As a general rule, try to avoid using more than three levels of nesting.



Good Programming Practice 4.4

The combination of **vertical spacing** before and after control statements and indentation of the bodies of control statements within the control-statement headers gives programs a two-dimensional appearance that greatly improves program readability.



Outline

fig04_02.c

```
1 /* Fig. 4.2: fig04_02.c
2  Counter-controlled repetition with the for statement */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int counter; /* define counter */
9
10    /* initialization, repetition condition, and increment
11       are all included in the for statement header. */
12    for ( counter = 1; counter <= 10; counter++ ) { ←
13        printf( "%d\n", counter );
14    } /* end for */
15
16    return 0; /* indicate program ended successfully */
17
18 } /* end function main */
```

for loop begins by setting **counter** to 1 and repeats while **counter** \leq 10. Each time the end of the loop is reached, **counter** is incremented by 1.



این تصویر توضیح می‌دهد که چگونه یک حلقه **for** در برنامه‌نویسی کار می‌کند. اجزای اصلی آن به شرح زیر است:

کلمه کلیدی: **for** که برای شروع حلقه استفاده می‌شود.

متغیر کنترلی: **int counter** که برای شمارش تکرارها استفاده می‌شود.

مقدار اولیه متغیر کنترلی: **counter = 1** که در ابتدا تنظیم می‌شود.

شرط ادامه حلقه: **counter <= 10** که تا زمانی که صادق باشد، حلقه ادامه پیدا می‌کند.

افزایش متغیر کنترلی: **counter++** که در هر تکرار مقدار متغیر را یک واحد افزایش می‌دهد.

این ساختار باعث می‌شود که حلقه ۱۰ بار اجرا شود و مقدار **counter** از ۱ تا ۱۰ افزایش یابد.

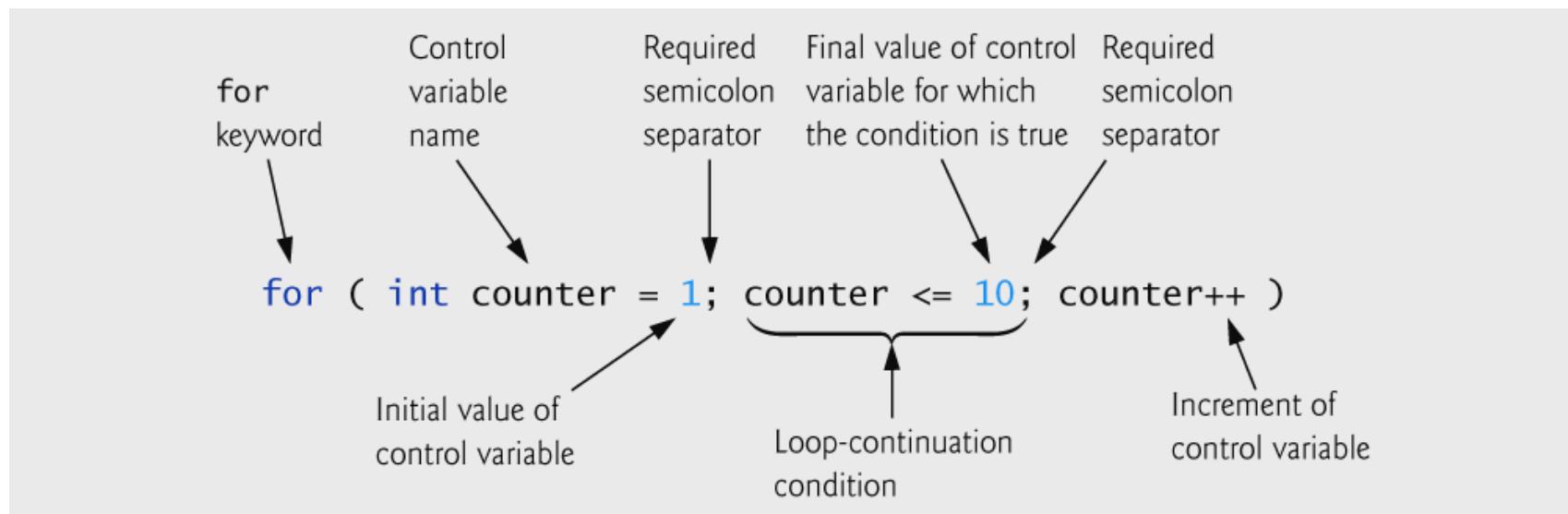


Fig. 4.3 : **for** statement header components.



Common Programming Error 4.2

**Using an incorrect relational operator or
using an incorrect initial or final value of a
loop counter in the condition of a `while` or
`for` statement can cause off-by-one errors.**



Error-Prevention Tip 4.2

Using the final value in the condition of a **while** or **for** statement and using the **\leq** relational operator will help avoid off-by-one errors. For a loop used to print the values 1 to 10, for example, the loop-continuation condition should be **counter \leq 10** rather than **counter $<$ 11** or **counter $<$ 10**.



استفاده از عملگر رابطه‌ای نادرست یا استفاده از مقدار اولیه یا نهایی نادرست شمارنده حلقه در شرط یک دستور **for** یا **while** می‌تواند منجر به خطا‌ی **off-by-one** شود.

استفاده از مقدار نهایی در شرط یک دستور **while** یا **for** و استفاده از عملگر رابطه‌ای $= <$ کمک می‌کند تا از خطا‌های **off-by-one** اجتناب شود. برای مثال، در یک حلقه که برای چاپ مقادیر ۱ تا ۱۰ استفاده می‌شود، شرط ادامه حلقه باید **10** باشد، نه **counter < 10** یا **counter < 11**.

به این ترتیب، مقدار نهایی **counter** برابر ۱۰ خواهد بود و همه مقادیر ۱ تا ۱۰ چاپ می‌شوند. استفاده از شرط نادرست ممکن است باعث چاپ مقادیر ۱ تا ۹ یا ۲ تا ۱۰ شود که خطا‌های **off-by-one** هستند.



4.4 for Repetition Statement

- Format when using **for** loops

```
for ( initialization; loopContinuationTest; increment )  
    statement
```

- Example:

```
for( int counter = 1; counter <= 10; counter++ )  
    printf( "%d\n", counter );
```

- Prints the integers from one to ten



4.4 for Repetition Statement

حلقه‌های `for` معمولاً می‌توانند به صورت حلقه‌های `while` بازنویسی شوند:

- For loops can usually be rewritten as while loops:

```
initialization;
while ( loopContinuationTest ) {
    statement;
    increment;
}
```

- Initialization and increment

- Can be comma-separated lists
- Example:

```
for (int i = 0, j = 0; j + i <= 10; j++,  
     i++)
    printf( "%d\n", j + i );
```

مقداردهی اولیه و افزایش:

می‌توانند به صورت لیست‌های جداشده با کاما نوشته شوند.

در این مثال، مقداردهی اولیه و افزایش متغیرها او زی به صورت لیست‌های جداگانه آمده‌اند. این دو متغیر همزمان در حلقه افزایش می‌یابند.



Software Engineering Observation 4.1

Place only expressions involving the control variables in the initialization and increment sections of a `for` statement. Manipulations of other variables should appear either before the loop (if they execute only once, like initialization statements) or in the loop body (if they execute once per repetition, like incrementing or decrementing statements).



Common Programming Error 4.3

**Using commas instead of semicolons in a
for header is a syntax error.**



Common Programming Error 4.4

Placing a semicolon immediately to the right of a `for` header makes the body of that `for` statement an empty statement. This is normally a logic error.



نکات مهم برای استفاده از دستور `:for`

- در بخش مقداردهی اولیه و افزایش / کاهش حلقه `for`, فقط عبارت‌های مربوط به متغیرهای کنترلی را قرار دهید.
- دستورات دیگر مانند مقداردهی اولیه یا تغییر متغیرهای غیرکنترلی باید قبل از حلقه (اگر فقط یک بار اجرا شوند) در بدنه حلقه (اگر در هر تکرار اجرا شوند) قرار گیرند.
- استفاده از کاما به جای سیمکولن در هدر حلقه `for` یک خطای نحوی است.
- قرار دادن سیمکولن بلافاصله پس از هدر حلقه `for`, بدنه حلقه را به یک دستور خالی تبدیل می‌کند. این معمولاً یک خطای منطقی است.
- به طور خلاصه، باید دقت کرد که ساختار حلقه `for` به درستی نوشته شود تا از بروز خطاهای نحوی و منطقی جلوگیری شود.



4.5 for Statement : Notes and Observations

■ Arithmetic expressions

- Initialization, loop-continuation, and increment can contain arithmetic expressions. If x equals 2 and y equals 10

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

■ Notes about the for statement:

- "Increment" may be negative (decrement)
- If the loop continuation condition is initially false
 - The body of the for statement is not performed
 - Control proceeds with the next statement after the for statement
- Control variable
 - Often printed or used inside for body, but not necessary



بیان‌های حسابی:

- مقداردهی اولیه، شرط ادامه حلقه و افزایش/کاهش می‌توانند شامل بیان‌های حسابی باشند.
- مثال: اگر $x=2$ و $y=10$ باشد، معادل است با $for (j=x; j<=4xy; j+=y/x)$

`for (j=2; j<=80; j+=5);`

یادداشت‌ها درباره دستور **for**:

- افزایش ممکن است منفی باشد (کاهش).
- اگر شرط ادامه حلقه در ابتدا نادرست باشد، بدنه حلقه اجرا نمی‌شود.
- کنترل به دستور بعدی پس از حلقه **for** منتقل می‌شود.
- متغیر کنترلی:

- معمولاً در بدنه حلقه چاپ یا استفاده می‌شود، اما لزومی ندارد.

- در مجموع، این اسلاید نکات مهمی درباره استفاده صحیح از دستور **for** در برنامه‌نویسی ارائه می‌دهد.



Error-Prevention Tip 4.3

Although the value of the control variable can be changed in the body of a `for` loop, this can lead to subtle errors. It is best not to change it.

گرچه مقدار متغیر کنترلی در بدن حلقه `for` قابل تغییر است،
این کار می‌تواند به خطاها نامحسوس منجر شود. بهتر است
که آن را تغییر ندهید.



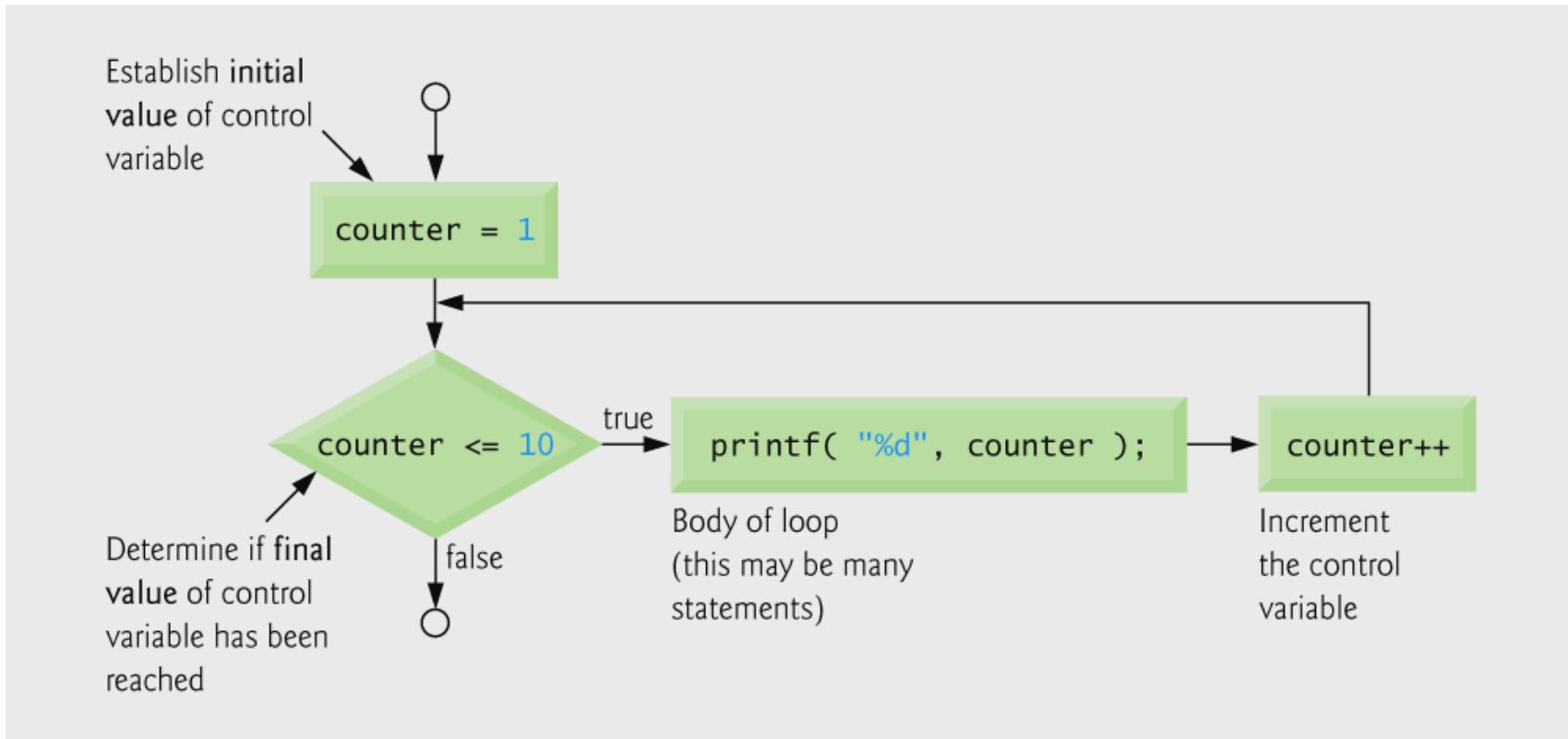


Fig. 4.4 | Flowcharting a typical **for** repetition statement.



Outline

fig04_05.c

```

1 /* Fig. 4.5: fig04_05.c
2   Summation with for */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int sum = 0; /* initialize sum */
9     int number; /* number to be added to sum */
10
11    for ( number = 2; number <= 100; number += 2 ) {
12        sum += number; /* add number to sum */ ←
13    } /* end for */
14
15    printf( "Sum is %d\n", sum ); /* output sum */
16
17    return 0; /* indicate program ended successfully */
18
19 } /* end function main */

```

Sum is 2550

Note that **number** has a different value
each time this statement is executed



Good Programming Practice 4.5

Although statements preceding a **for** and statements in the body of a **for** can often be merged into the **for** header, avoid doing so because it makes the program more difficult to read.



Good Programming Practice 4.6

Limit the size of control-statement headers to a single line if possible.

هرچند می‌توان دستورات قبل از حلقه `for` و دستورات داخل بدنه آن را اغلب در بخش سرآیند `for` ادغام کرد، از این کار پرهیز کنید زیرا باعث دشوارتر شدن خوانایی برنامه می‌شود.

در صورت امکان، اندازه‌ی سرآیند دستورات کنترل را به یک خط محدود کنید.



Outline

```

1 /* Fig. 4.6: fig04_06.c
2   Calculating compound interest */
3 #include <stdio.h>
4 #include <math.h> ← additional header
5
6 /* function main begins program execution */
7 int main( void )
8 {
9     double amount;           /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;       /* annual interest rate */
12    int year;               /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20        /* calculate new amount for specified year */
21        amount = principal * pow( 1.0 + rate, year ); ←
22
23        /* output one table row */
24        printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
27    return 0; /* indicate program ended successfully */
28
29 } /* end function main */

```

fig04_06.c

(1 of 2)

pow function calculates the value of the first argument raised to the power of the second argument



Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

Outline

fig04_06.c
(2 of 2)



Error-Prevention Tip 4.4

Do not use variables of type `float` or `double` to perform monetary calculations. The imprecision of floating-point numbers can cause errors that will result in incorrect monetary values.

از متغیرهای نوع `float` یا `double` برای انجام محاسبات پولی استفاده نکنید. عدم دقت اعداد اعشاری ممیزشناور می‌تواند موجب بروز خطاهای شود که به مقادیر پولی نادرست منتهی می‌گردد.



4.7 switch Multiple-Selection Statement

زمانی مفید است که یک متغیر یا عبارت برای تمام مقادیری که می‌تواند اختیار کند آزمایش شود و اقدامات متفاوتی انجام گیرد.

- **switch**

- Useful when a variable or expression is tested for all the values it can assume and different actions are taken

- **Format**

- Series of case labels and an optional default case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
}
```

- **break;** exits from statement

دنباله‌ای از برجسب‌های case و یک مورد اختیاری default:

دستور break; از عبارت سوئیچ خارج می‌شود.



Outline

fig04_07.c

(1 of 4)

```

1 /* Fig. 4.7: fig04_07.c
2   Counting letter grades */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int grade;      /* one grade */
9     int aCount = 0; /* number of As */
10    int bCount = 0; /* number of Bs */
11    int cCount = 0; /* number of Cs */
12    int dCount = 0; /* number of Ds */
13    int fCount = 0; /* number of Fs */
14
15    printf( "Enter the letter grades.\n" );
16    printf( "Enter the EOF character to end input.\n" );
17
18    /* Loop until user types end-of-file key sequence */
19    while ( ( grade = getchar() ) != EOF ) { ←
20
21        /* determine which grade was input */
22        switch ( grade ) { /* switch nested in while */ ←
23
24            case 'A': /* grade was uppercase A */ ←
25            case 'a': /* or lowercase a */ ←
26                ++aCount; /* increment aCount */
27                break; /* necessary to exit switch */ ←
28

```

EOF stands for “end of file;” this character varies from system to system

switch statement checks each of its nested **cases** for a match

break statement makes program skip to end of **switch**



Outline

```
29 case 'B': /* grade was uppercase B */
30 case 'b': /* or lowercase b */
31     ++bCount; /* increment bCount */
32     break; /* exit switch */

33

34 case 'C': /* grade was uppercase C */
35 case 'c': /* or lowercase c */
36     ++cCount; /* increment cCount */
37     break; /* exit switch */

38

39 case 'D': /* grade was uppercase D */
40 case 'd': /* or lowercase d */
41     ++dCount; /* increment dCount */
42     break; /* exit switch */

43

44 case 'F': /* grade was uppercase F */
45 case 'f': /* or lowercase f */
46     ++fCount; /* increment fCount */
47     break; /* exit switch */

48

49 case '\n': /* ignore newlines, */
50 case '\t': /* tabs, */
51 case ' ': /* and spaces in input */
52     break; /* exit switch */

53
```

fig04_07.c

(2 of 4)



```
44     default: /* catch all other characters */
45         printf( "Incorrect letter grade entered." );
46         printf( " Enter a new grade.\n" );
47         break; /* optional; will exit switch anyway */
48     } /* end switch */
49
50 } /* end while */
51
52 /* output summary of results */
53 printf( "\nTotals for each letter grade are:\n" );
54 printf( "A: %d\n", aCount ); /* display number of A grades */
55 printf( "B: %d\n", bCount ); /* display number of B grades */
56 printf( "C: %d\n", cCount ); /* display number of C grades */
57 printf( "D: %d\n", dCount ); /* display number of D grades */
58 printf( "F: %d\n", fCount ); /* display number of F grades */
59
60     return 0; /* indicate program ended successfully */
61
62 } /* end function main */
```

Outline

default case occurs if none of the **cases** are matched

fig04_07.c

(3 of 4)



Outline

fig04_07.c

(4 of 4)

Enter the letter grades.

Enter the EOF character to end input.

a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:

A: 3
B: 2
C: 3
D: 2
F: 1



Portability Tip 4.1

The keystroke combinations for entering EOF (end of file) are system dependent.

ترکیب کلیدهای مورد نیاز برای وارد کردن EOF(پایان فایل) به سیستم وابسته است.



Portability Tip 4.2

Testing for the symbolic constant EOF rather than -1 makes programs more portable. The C standard states that EOF is a negative integral value (but not necessarily -1). Thus, EOF could have different values on different systems.

آزمایش برای ثابت نمادین EOF به جای عدد -1 ، برنامه‌ها را قابل حمل‌تر می‌کند. استاندارد C بیان می‌کند که EOF یک مقدار عددی منفی است (اما نه لزوماً -1). بنابراین EOF می‌تواند در سیستم‌های مختلف مقادیر متفاوتی داشته باشد.



Common Programming Error 4.5

Forgetting a `break` statement when one is needed in a `switch` statement is a logic error.

فراموش کردن دستور `break` در جایی که در عبارت `switch`` مورد نیاز است، یک خطای منطقی محسوب می‌شود.



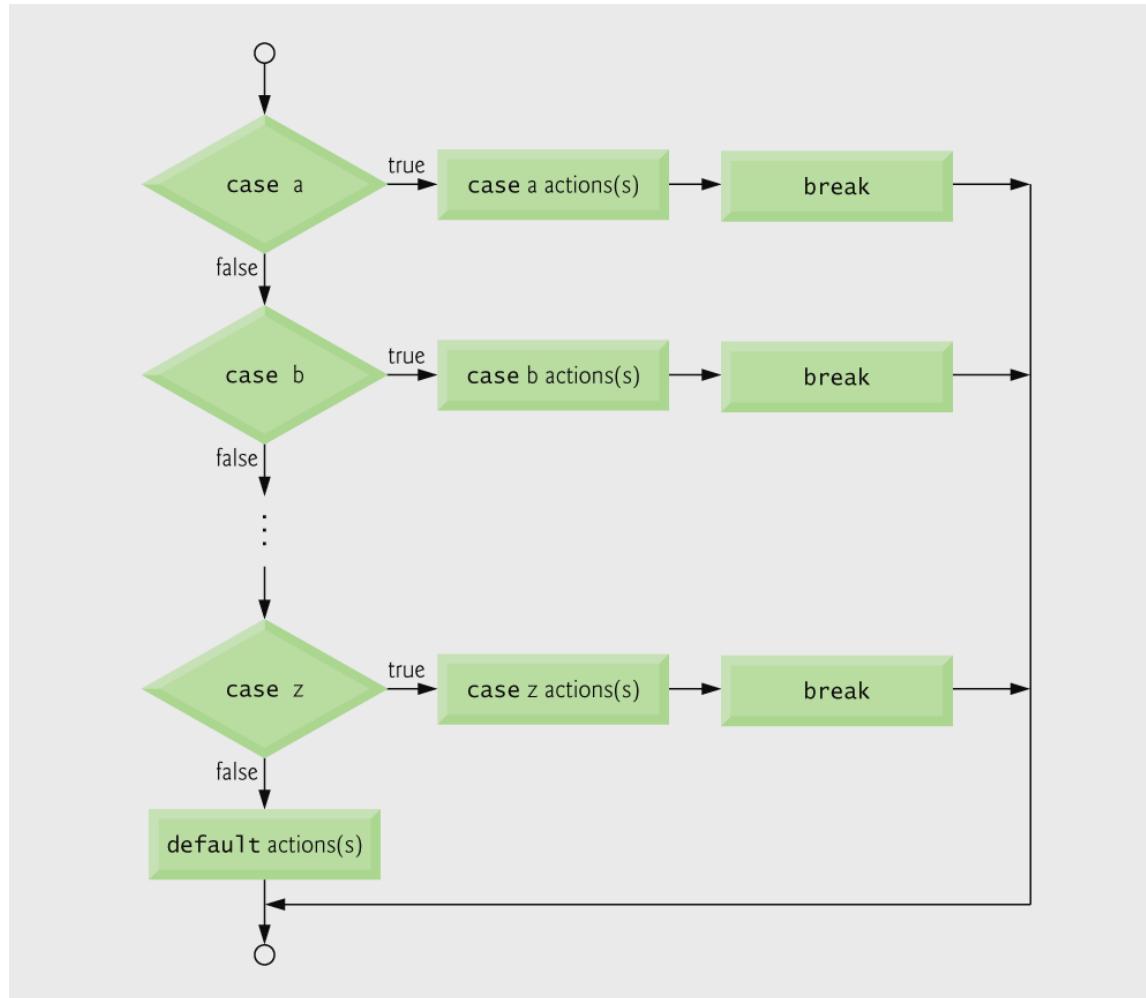


Fig. 4.8 | switch multiple-selection statement with breaks.



Good Programming Practice 4.7

Provide a **default** case in **switch** statements. Cases not explicitly tested in a **switch** are ignored. The **default** case helps prevent this by focusing the programmer on the need to process exceptional conditions. There are situations in which no **default** processing is needed.

یک مورد **default** در دستورات **switch** در نظر بگیرید. مواردی که به صراحت در **switch** آزمایش نشده‌اند، نادیده گرفته می‌شوند. مورد **default** با تمرکز دادن برنامه‌نویس بر نیاز به پردازش شرایط استثنایی، به جلوگیری از این مسئله کمک می‌کند. موقعی وجود دارد که هیچ پردازش پیش‌فرضی مورد نیاز نیست.



Good Programming Practice 4.8

Although the `case` clauses and the `default` case clause in a `switch` statement can occur in any order, it is considered good programming practice to place the `default` clause last.

هرچند بندهای `case` و بند `default` در دستور `switch` می‌توانند به هر ترتیبی قرار گیرند، اما قرار دادن بند `default` در انتهای بند `case`‌ها به عنوان یک روش برنامه‌نویسی خوب در نظر گرفته می‌شود.



Good Programming Practice 4.9

In a **switch** statement when the **default** clause is listed last, the **break** statement is not required. But some programmers include this **break** for clarity and symmetry with other **cases**.

در یک دستور **switch** وقتی بند **default** در انتهای فهرست شده باشد، دستور **break** الزامی نیست. اما برخی برنامهنویسان برای وضوح و تقارن با سایر موارد، این **break** را نیز قرار می‌دهند.



Common Programming Error 4.6

Not processing newline characters in the input when reading characters one at a time can cause logic errors.

عدم پردازش کاراکترهای خط جدید (`newline`) در ورودی هنگام خواندن کاراکترها به صورت تکی، می‌تواند باعث بروز خطاهای منطقی شود.



Error-Prevention Tip 4.5

Remember to provide processing capabilities for newline (and possibly other white-space) characters in the input when processing characters one at a time.

به یاد داشته باشید هنگام پردازش کاراکترها به صورت تکی، قابلیت‌های پردازشی برای کاراکترهای خط جدید (و احتمالاً سایر کاراکترهای فاصله سفید) در ورودی فراهم کنید.



4.8 do...while Repetition Statement

- The **do...while** repetition statement

- Similar to the **while** structure
- Condition for repetition only tested after the body of the loop is performed
 - All actions are performed at least once
- Format:

```
do {  
    statement;  
} while ( condition );
```

دستور تکرار **do...while**

شبیه به ساختار **while** است .

شرط تکرار تنها پس از اجرای بدنه حلقه آزمایش می شود .

همه عملیات حداقل یک بار انجام می گیرند.



4.8 do...while Repetition Statement

- Example (letting counter = 1):

```
do {  
    printf( "%d  ", counter );  
} while ( ++counter <= 10 );
```

- Prints the integers from 1 to 10



Good Programming Practice 4.10

Some programmers always include braces in a `do...while` statement even if the braces are not necessary. This helps eliminate ambiguity between the `do...while` statement containing one statement and the `while` statement.

برخی از برنامهنویسان همیشه در دستور `do...while` آکولاد قرار می‌دهند، حتی اگر آکولاد لازم نباشد. این کار به رفع ابهام بین دستور `do...while` که شامل یک دستور است و دستور `while` کمک می‌کند.



Common Programming Error 4.7

Infinite loops are caused when the loop-continuation condition in a `while`, `for` or `do...while` statement never becomes false. To prevent this, make sure there is not a semicolon immediately after the header of a `while` or `for` statement. In a counter-controlled loop, make sure the control variable is incremented (or decremented) in the loop. In a sentinel-controlled loop, make sure the sentinel value is eventually input.

حلقه‌ای بی‌نهایت هنگامی رخ می‌دهند که شرط ادامه حلقه در دستورات `while`, `for` یا `do...while` مهرگز `false` نشود. برای جلوگیری از این اتفاق، مطمئن شوید که بلافاصله پس از سرآیند دستور `while` یا `for`، نقطه‌ویرگول قرار نداده باشد. در یک حلقه کنترل شده با شمارنده، مطمئن شوید که متغیر کنترل در حلقه افزایش (یا کاهش) می‌یابد. در یک حلقه کنترل شده با نگهبان، مطمئن شوید که مقدار نگهبان در نهایت وارد می‌شود.



Outline

fig04_09.c

```

1 /* Fig. 4.9: fig04_09.c
2  Using the do/while repetition statement */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int counter = 1; /* initialize counter */
9
10    do {
11        printf( "%d ", counter ); /* display counter */
12    } while ( ++counter <= 10 ); /* end do...while */ ←
13
14    return 0; /* indicate program ended successfully */
15
16 } /* end function main */

```

1 2 3 4 5 6 7 8 9 10

increments **counter** then checks if it is less than or equal to 10



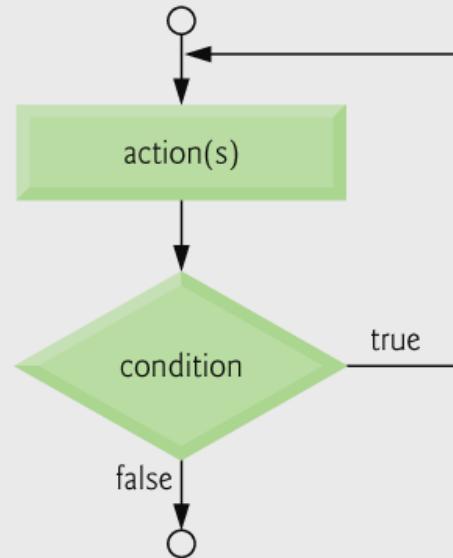


Fig. 4.10 | Flowcharting the **do...while** repetition statement.



4.9 break and continue Statements

■ **break**

- Causes immediate exit from a **while**, **for**, **do...while** or **switch** statement
- Program execution continues with the first statement after the structure
- Common uses of the **break** statement
 - Escape early from a loop
 - Skip the remainder of a **switch** statement

دستور **break** باعث خروج فوری از ساختارهای **do...while** , **for..while** **switch** یا **switch** می شود.
اجرای برنامه با اولین دستور پس از ساختار ادامه می یابد.

موارد رایج استفاده از دستور **:break**

- خروج زودهنگام از یک حلقه
- رد کردن بخش باقیمانده یک دستور **switch**



Outline

fig04_11.c

```

1 /* Fig. 4.11: fig04_11.c
2  Using the break statement in a for statement */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int x; /* counter */
9
10    /* Loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13        /* if x is 5, terminate loop */
14        if ( x == 5 ) {
15            break; /* break loop only if x is 5 */ ←
16        } /* end if */
17
18        printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */

```

break immediately ends **for** loop

1 2 3 4
Broke out of loop at x == 5



4.9 break and continue Statements

■ continue

- Skips the remaining statements in the body of a **while**, **for** or **do...while** statement
 - Proceeds with the next iteration of the loop
- **while** and **do...while**
 - Loop-continuation test is evaluated immediately after the **continue** statement is executed
- **for**
 - Increment expression is executed, then the loop-continuation test is evaluated

دستور continue

دستورات باقیمانده در بدن ساختارهای do...while، while، for و به تکرار بعدی حلقه می‌رود.

در ساختارهای while و do...while:

بلافاصله پس از اجرای دستور continue، شرط ادامه حلقه ارزیابی می‌شود.

در ساختار for:

عبارت افزایش (increment) اجرا می‌شود و سپس شرط ادامه حلقه ارزیابی می‌گردد.



Outline

fig04_12.c

```

1 /* Fig. 4.12: fig04_12.c
2  Using the continue statement in a for statement */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int x; /* counter */
9
10    /* Loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13        /* if x is 5, continue with next iteration of loop */
14        if ( x == 5 ) {
15            continue; /* skip remaining code in loop body */←
16        } /* end if */
17
18        printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */

```

1 2 3 4 6 7 8 9 10

Used continue to skip printing the value 5

continue skips to end of **for** loop and performs next iteration



Software Engineering Observation 4.2

Some programmers feel that `break` and `continue` violate the norms of structured programming. Because the effects of these statements can be achieved by structured programming techniques we will soon learn, these programmers do not use `break` and `continue`.

برخی برنامهنویسان معتقدند که دستورات `break` و `continue` هنجرهای برنامهنویسی ساختاریافته را نقض می‌کنند. از آنجایی که اثرات این دستورات را می‌توان با تکنیک‌های برنامهنویسی ساختاریافته‌ای که به زودی خواهیم آموخت به دست آورد، این برنامهنویسان از `break` و `continue` استفاده نمی‌کنند.



Performance Tip 4.1

The `break` and `continue` statements, when used properly, perform faster than the corresponding structured techniques that we will soon learn.

دستورات `break` و `continue`، در صورت استفاده صحیح، عملکرد سریع‌تری نسبت به تکنیک‌های ساختاریافته معادل که به زودی خواهیم آموخت ارائه می‌دهند.



Software Engineering Observation 4.3

There is a tension between achieving quality software engineering and achieving the best-performing software. Often one of these goals is achieved at the expense of the other.

بین دستیابی به مهندسی نرم افزار با کیفیت و دستیابی به نرم افزار با بهترین عملکرد، تنشی وجود دارد. اغلب یکی از این اهداف به قیمت از دست دادن دیگری محقق می شود.



4.10 Logical Operators

- **&& (logical AND)**
 - Returns **true** if both conditions are **true**
- **|| (logical OR)**
 - Returns **true** if either of its conditions are **true**
- **! (logical NOT, logical negation)**
 - Reverses the truth/falsity of its condition
 - Unary operator, has one operand
- **Useful as conditions in loops**

<u>Expression</u>	<u>Result</u>
<code>true && false</code>	<code>false</code>
<code>true false</code>	<code>true</code>
<code>!false</code>	<code>true</code>



expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Fig. 4.13 | Truth table for the **&&** (logical AND) operator.



expression1	expression2	expression1 expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Fig. 4.14 | Truth table for the logical OR (||) operator.



expression	$!\text{expression}$
0	1
nonzero	0

Fig. 4.15 | Truth table for operator $!$ (logical negation).



Performance Tip 4.2

In expressions using operator `&&`, make the condition that is most likely to be false the leftmost condition. In expressions using operator `||`, make the condition that is most likely to be true the leftmost condition. This can reduce a program's execution time.

در عبارت‌هایی که از عملگر `&&` استفاده می‌کنند، شرطی که احتمال `false` بودن آن بیشتر است را در سمت‌چپ‌ترین موقعیت قرار دهید. در عبارت‌هایی که از عملگر `||` استفاده می‌کنند، شرطی که احتمال `true` بودن آن بیشتر است را در سمت‌چپ‌ترین موقعیت قرار دهید. این کار می‌تواند زمان اجرای برنامه را کاهش دهد.



Operators	Associativity	Type
<code>++ (postfix)</code> <code>-- (postfix)</code>	right to left	postfix
<code>+</code> <code>-</code> <code>!</code> <code>++ (prefix)</code> <code>-- (prefix)</code> <code>(type)</code>	right to left	unary
<code>*</code> <code>/</code> <code>%</code>	left to right	multiplicative
<code>+</code> <code>-</code>	left to right	additive
<code><</code> <code><=</code> <code>></code> <code>>=</code>	left to right	relational
<code>==</code> <code>!=</code>	left to right	equality
<code>&&</code>	left to right	logical AND
<code> </code>	left to right	logical OR
<code>?:</code>	right to left	conditional
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	right to left	assignment
<code>,</code>	left to right	comma

Fig. 4.16 | Operator precedence and associativity.



4.11 Confusing Equality (==) and Assignment (=) Operators

این نوع خطا معمولاً باعث بروز خطاها نحوی نمی‌شود.

- **Dangerous error** هر عبارتی که مقداری تولید کند می‌تواند در ساختارهای کنترلی استفاده شود. مقادیر غیرصفر به عنوان **true** و مقادیر صفر به عنوان **false** در نظر گرفته می‌شوند.

- Does not ordinarily cause syntax errors
- Any expression that produces a value can be used in control structures
- Nonzero values are **true**, zero values are **false**
- Example using ==:

```
if ( payCode == 4 )
    printf( "You get a bonus!\n" );
```

- Checks **payCode**, if it is 4 then a bonus is awarded

این کد **payCode** را روی ۴ تنظیم می‌کند.

از آنجا که ۴ غیرصفر است، عبارت شرط **true** ارزیابی می‌شود و بدون توجه به مقدار قبلی **payCode**، پاداش تعلق می‌گیرد.

این یک خطای منطقی است، نه خطای نحوی.



4.11 Confusing Equality (==) and Assignment (=) Operators

- Example, replacing == with =:

```
if ( payCode = 4 )  
    printf( "You get a bonus!\n" );
```

This sets payCode to 4

4 is nonzero, so expression is true, and bonus awarded no matter what the payCode was

- Logic error, not a syntax error



Common Programming Error 4.8

Using operator == for assignment or using operator = for equality is a logic error.

استفاده از عملگر == برای انتساب یا استفاده از عملگر = برای برابری،
یک خطای منطقی محسوب می‌شود.



4.11 Confusing Equality (==) and Assignment (=) Operators

■ lvalues

- Expressions that can appear on the left side of an equation
- Their values can be changed, such as variable names

- $x = 4;$

عبارت‌هایی که می‌توانند در سمت چپ معادله ظاهر شوند.
مقادیر آنها قابل تغییر است، مانند نام متغیرها.

■ rvalues

- Expressions that can only appear on the right side of an equation
- Constants, such as numbers
 - Cannot write $4 = x;$
 - Must write $x = 4;$
- lvalues can be used as rvalues, but not vice versa
 - $y = x;$

value‌ها می‌توانند به عنوان rvalue استفاده شوند، اما برعکس آن ممکن نیست.



Good Programming Practice 4.11

When an equality expression has a variable and a constant, as in `x == 1`, some programmers prefer to write the expression with the constant on the left and the variable name on the right (e.g. `1 == x` as protection against the logic error that occurs when you accidentally replace operator `==` with `=`.

هنگامی که یک عبارت برابری شامل یک متغیر و یک ثابت است (مانند `1 == x`، برخی برنامهنویسان ترجیح می‌دهند عبارت را با ثابت در سمت چپ و نام متغیر در سمت راست بنویسند (مثلاً `1 == x`) تا در برابر خطای منطقی ناشی از جایگزینی تصادفی عملگر `==` با `=` محافظت شود.



Error-Prevention Tip 4.6

After you write a program, text search it for every = and check that it is being used properly.

پس از نوشتن برنامه، آن را برای هر علامت = جستجو کنید و مطمئن شوید که به درستی استفاده شده است.



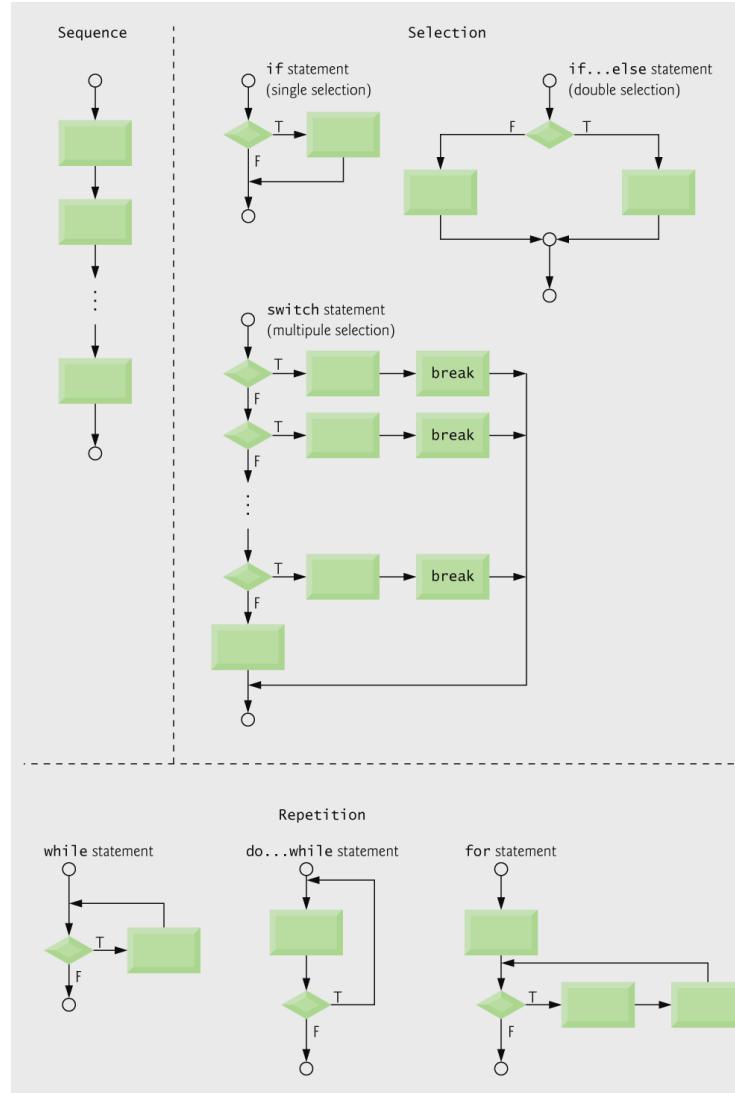


Fig. 4.17 | C's single-entry/single-exit sequence, selection and repetition statements.



4.12 Structured Programming Summary

- **Structured programming**

- Easier than unstructured programs to understand, test, debug and, modify programs

درک، آزمون، اشکال‌زدایی و اصلاح برنامه‌های ساختار یافته آسان‌تر از برنامه‌های غیرساختار یافته است.



قوانين تشکیل برنامه‌های ساخت‌یافته

۱) با "ساده‌ترین فلوچارت" آغاز کنید (شکل ۴.۱۹).

۲) هر مستطیل (عملیات) را می‌توان با دو مستطیل (عملیات) به صورت متوالی جایگزین کرد.

۳) هر مستطیل (عملیات) را می‌توان با هر دستور کنترلی جایگزین کرد (دنباله، `for`, `do...while`, `while`, `switch`, `if...else`)

۴) قوانین ۲ و ۳ را می‌توان به هر تعداد و به هر ترتیبی اعمال کرد.

Rules for Forming Structured Programs

- 1) Begin with the “simplest flowchart” (Fig. 4.19).
- 2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence.
- 3) Any rectangle (action) can be replaced by any control statement (sequence, `if`, `if...else`, `switch`, `while`, `do...while` or `for`).
- 4) Rules 2 and 3 may be applied as often as you like and in any order.

Fig. 4.18 | Rules for forming structured programs.



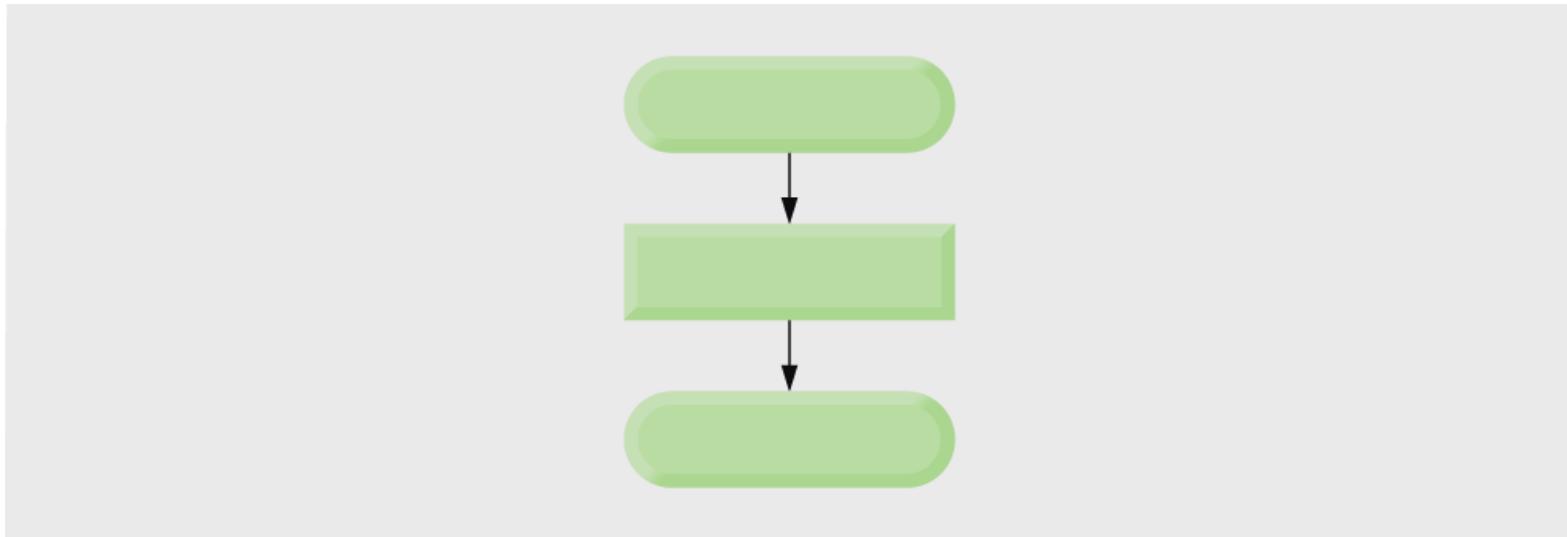


Fig. 4.19 | Simplest flowchart.



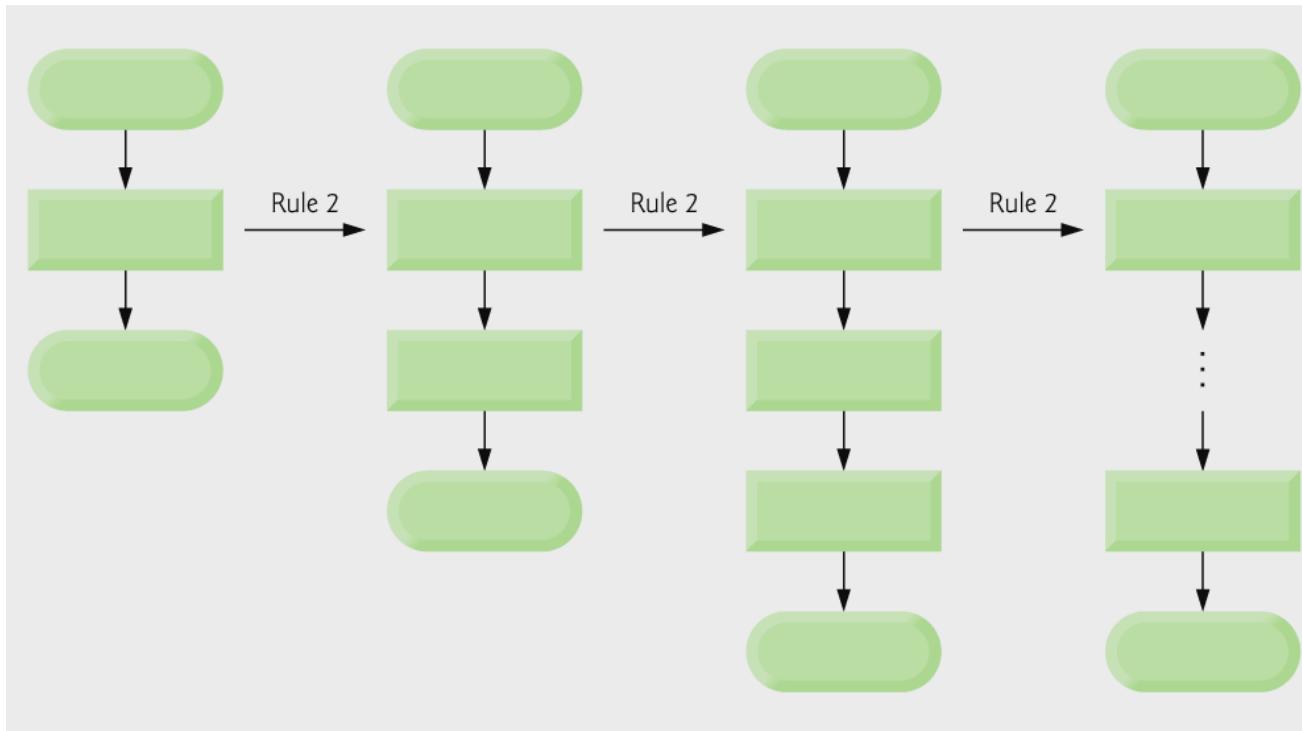


Fig. 4.20 | Repeatedly applying rule 2 of Fig. 4.18 to the simplest flowchart.



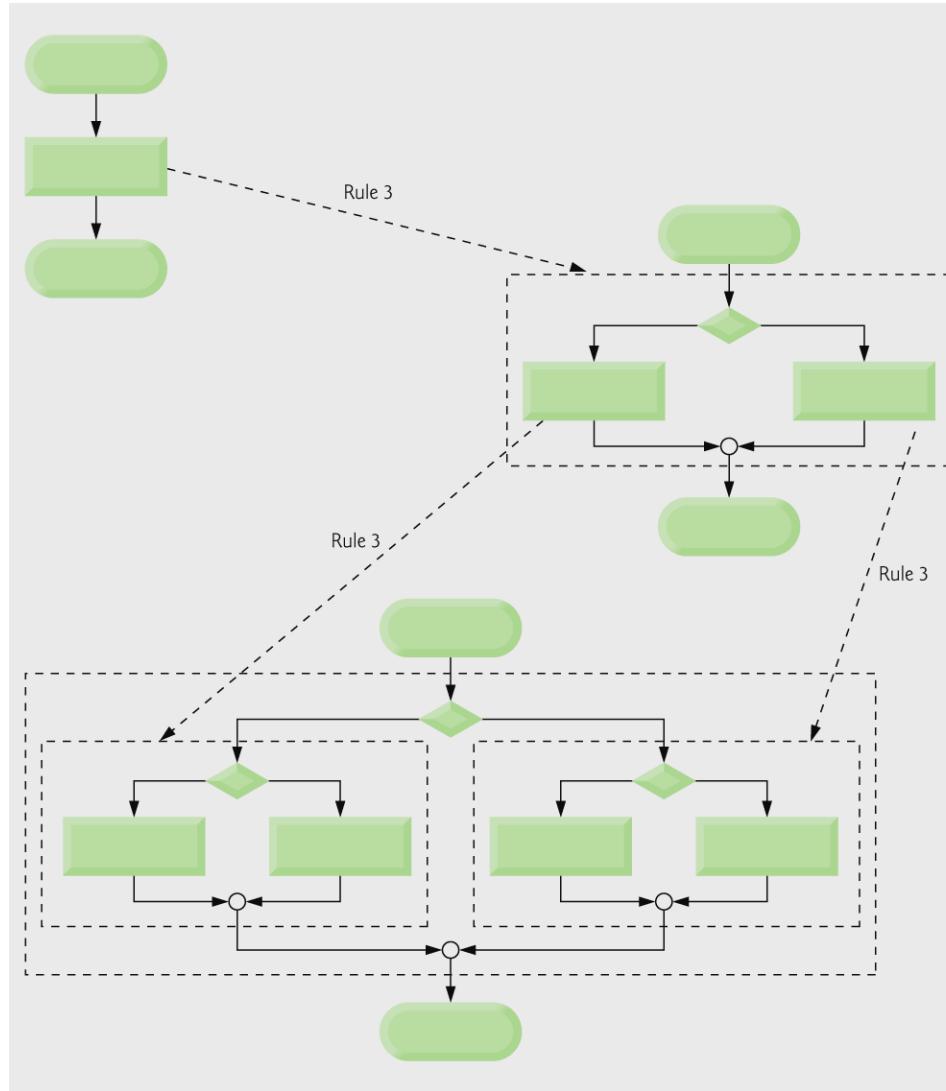


Fig. 4.21 | Applying rule 3 of Fig. 4.18 to the simplest flowchart.



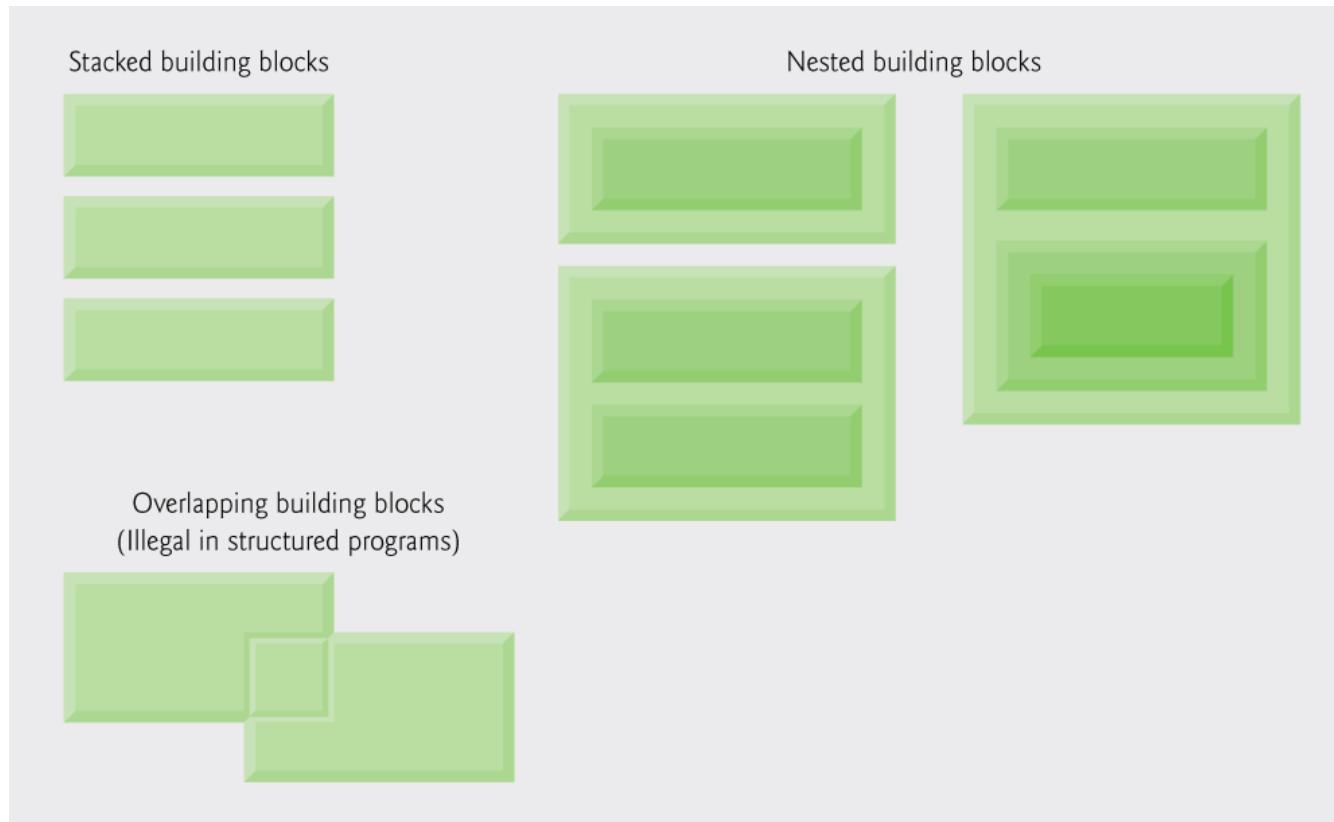


Fig. 4.22 | Stacked, nested and overlapped building blocks.

پشته ای، تودرتو و همپوشان



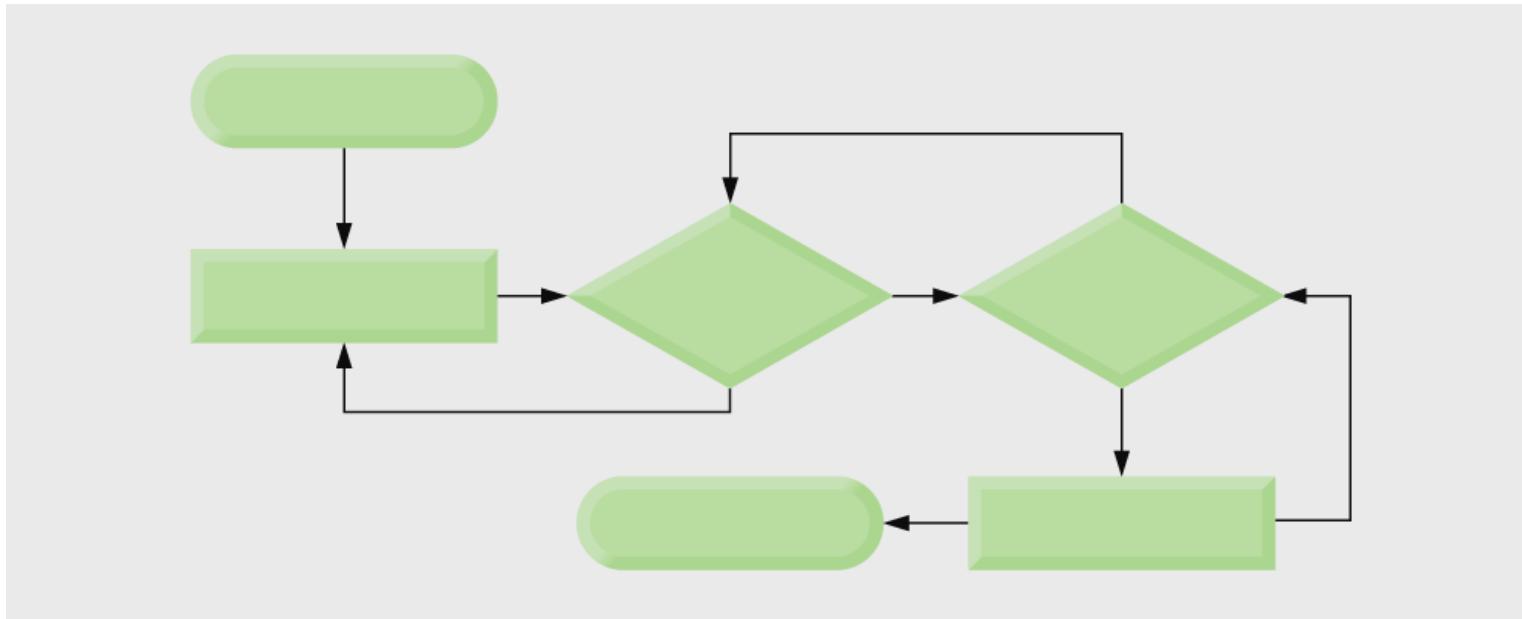


Fig. 4.23 | An unstructured flowchart.



4.12 Structured Programming Summary

- All programs can be broken down into 3 controls
 - Sequence – handled automatically by compiler
 - Selection – if, if...else or switch
 - Repetition – while, do...while or for
 - Can only be combined in two ways
 - Nesting (rule 3)
 - Stacking (rule 2)
 - Any selection can be rewritten as an if statement, and any repetition can be rewritten as a while statement



برنامه‌نویسی ساختار یافته

همه برنامه‌ها را می‌توان به ۳ ساختار کنترلی تجزیه کرد:

۱. ترتیب (Sequence): که به طور خودکار توسط کامپایلر مدیریت می‌شود.
۲. انتخاب (Selection): شامل `if...else` یا `switch`.
۳. تکرار (Repetition): شامل `for`, `do...while` یا `while`.

این ساختارها فقط به دو طریق می‌توانند ترکیب شوند:

تودرتو کردن (Nesting): قاعده ۳

چیدن پشت سر هم (Stacking): قاعده ۲

هر ساختار انتخاب را می‌توان به صورت یک دستور `if` بازنویسی کرد و هر ساختار تکرار را می‌توان به صورت یک دستور `while` بازنویسی کرد.

