

Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Introduction to C Programming

Data Type and Variables

Data Types

- C has a number of basic **data types**.
- Each have specific uses and advantages, depending on the application.
- In general, the data types can be divided into two categories; **Integer types** and **Floating Point types**.

Data Types

- Integer types:

Name	Size (in bits, on x86)	Range	Notes
bool	8 (top 7 bits are ignored)	0 or 1	C++ only
char	8	-128 to 127(signed) or 0-255(unsigned)	standard issue "byte"
short	16	-32768 to 32767(signed) or 0-65536(unsigned)	just like a char , only twice as large
int	32	-2147483648 to 2147483647(signed) or 0-4294967296(unsigned)	standard-issue integer number type
long	32 (can be 64 on other architectures)	same as int	ditto
long long	64 (this is a non-standard GNU extension)	-9223372036854775808 to 9223372036854775807(signed) or 0-18446744073709551616(unsigned)	For very huge integers

- Floating Point types:

Name	Size (in bits, on x86)	Range	Notes
float	32	+/- 1.4023×10^{-45} to $3.4028 \times 10^{+38}$	general purpose real-number
double	64	+/- 4.9406×10^{-324} to 1.7977×10^{308}	higher-precision real number
long double	96 (this is a non-standard GNU extention)	???	For numbers with very large ranges and high precision

Data Types

- Example: [datasize.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Size of Data Type (bytes)\n");
6     printf("short\t\t\t%lu\n", sizeof(short));
7     printf("int\t\t\t%lu\n", sizeof(int));
8     printf("long\t\t\t%lu\n", sizeof(long));
9     printf("long long\t\t\t%lu\n", sizeof(long long));
10    printf("float\t\t\t%lu\n", sizeof(float));
11    printf("double\t\t\t%lu\n", sizeof(double));
12    printf("long double\t\t\t%lu\n", sizeof(long double));
13    printf("char\t\t\t%lu\n", sizeof(char));
14
15    return 0;
16 }
```

Data Types

- Example: [datasize.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Size of Data Type (bytes)\n");
6     printf("short\t\t\t%lu\n", sizeof(short));
7     printf("int\t\t\t%lu\n", sizeof(int));
8     printf("long\t\t\t%lu\n", sizeof(long));
9     printf("long long\t\t\t%lu\n", sizeof(long long));
10    printf("float\t\t\t%lu\n", sizeof(float));
11    printf("double\t\t\t%lu\n", sizeof(double));
12    printf("long double\t\t\t%lu\n", sizeof(long double));
13    printf("char\t\t\t%lu\n", sizeof(char));
14
15    return 0;
16 }
```

Size of Data Type (bytes)		
short	2	
int	4	
long	8	
long long		8
float	4	
double	8	
long double	16	
char	1	

Overflow

- In computer programming, **overflow** occurs when an arithmetic operation attempts to create a numeric value that is larger than can be represented within the available storage space.

Overflow

- Example: [overflow.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     short aInt = 32767;
6
7     printf("%d\n", aInt);
8
9     aInt = aInt + 1;
10    printf("%d\n", aInt);
11
12    aInt = aInt + 1;
13    printf("%d\n", aInt);
14
15    return 0;
16 }
```

Overflow

- Example: [overflow.c](#)


```
1 #include <stdio.h>
2
3 int main(void) {
4
5     short aInt = 32767;
6
7     printf("%d\n", aInt);
8
9     aInt = aInt + 1;
10    printf("%d\n", aInt);
11
12    aInt = aInt + 1;
13    printf("%d\n", aInt);
14
15    return 0;
16 }
```

```
32767
-32768
-32767
```

Overflow

- Example: [overflow.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     short aInt = 32767;
6
7     printf("%d\n", aInt);
8
9     aInt = aInt + 1;
10    printf("%d\n", aInt);
11
12    aInt = aInt + 1;
13    printf("%d\n", aInt);
14
15    return 0;
16 }
```



A diagram illustrating integer overflow. It shows a vertical stack of three numbers: 32767, -32768, and -32767. The numbers -32768 and -32767 are enclosed in a red rounded rectangle, indicating the result of an overflow from the initial value 32767.

32767
-32768
-32767

printf format specifier

%c	以字元 方式輸出
%d	10 進位整數輸出
%o	以8進 位整數方式輸出
%u	無號整數輸出
%x, %X	將整 數以16進位方式輸出
%f	浮點 數輸出
%e, %E	使用科學記號顯示浮點數
%g, %G	浮點數輸出，取%f或%e（%f或%E），看哪個表示精簡
%%	顯示 %
%s	字串輸出

printf format specifier

- Example: [printfformat.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Show char: %c\n", 'A');
6     printf("Show the ascii code of the char: %d\n", 'A');
7     printf("Show the encode of char: %c\n", 65);....
8     printf("Show decimal number: %d\n", 15);
9     printf("Show octal number: %o\n", 15);
10    printf("Show hexadecimal number: %x\n", 15);
11    printf("Show scientific notation: %E\n", 0.001234);....
12    printf("Show scientific notation: %e\n", 0.001234);....
13
14    printf("example:%10.2f\n", 19.234);
15    printf("example:%-10.2f\n", 19.234);
16
17    return 0;
18 }
```

printf format specifier

- Example: [printfformat.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Show char: %c\n", 'A');
6     printf("Show the acsii code of the char: %d\n", 'A');
7     printf("Show the encode of char: %c\n", 65);....
8     printf("Show decimal number: %d\n", 15);
9     printf("Show octal number: %o\n", 15);
10    printf("Show hexadecimal number: %x\n", 15);
11    printf("Show scientific notation: %E\n", 0.001234);....
12    printf("Show scientific notation: %e\n", 0.001234);....
13
14    printf("example:%10.2f\n", 19.234);
15    printf("example:%-10.2f\n", 19.234);
16
17    return 0;
18 }
```

```
Show char: A
Show the acsii code of the char: 65
Show the encode of char: A
Show decimal number: 15
Show octal number: 17
Show hexadecimal number: f
Show scientific notation: 1.234000E-03
Show scientific notation: 1.234000e-03
example:      19.23
example:19.23
```

printf format specifier

- Example: [printfformat.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf("Show char: %c\n", 'A');
6     printf("Show the acsii code of the char: %d\n", 'A');
7     printf("Show the encode of char: %c\n", 65);....
8     printf("Show decimal number: %d\n", 15);
9     printf("Show octal number: %o\n", 15);
10    printf("Show hexadecimal number: %x\n", 15);
11    printf("Show scientific notation: %E\n", 0.001234);....
12    printf("Show scientific notation: %e\n", 0.001234);....
13
14    printf("example:%10.2f\n", 19.234);
15    printf("example:%-10.2f\n", 19.234);
16
17    return 0;
18 }
```

```
Show char: A
Show the acsii code of the char: 65
Show the encode of char: A
Show decimal number: 15
Show octal number: 17
Show hexadecimal number: f
Show scientific notation: 1.234000E-03
Show scientific notation: 1.234000e-03
example:      19.23
example:19.23
```

scanf function

- **scanf** accepts multiple inputs
- must specify the accepted format

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

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


scanf function

- Example: [scanfformat.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int number1, number2;
5
6     printf("Please input two numbers (use space for separation): ");
7     scanf("%d %d", &number1, &number2);
8     printf("Your input numbers: %d %d\n", number1, number2);
9
10    printf("Please input two numbers (use - for separation): ");
11    scanf("%d-%d", &number1, &number2);
12    printf("Your input numbers: %d-%d\n", number1, number2);
13
14    return 0;
15 }
```

scanf function

- Example: [scanfformat.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int number1, number2;
5
6     printf("Please input two numbers (use space for separation): ");
7     scanf("%d %d", &number1, &number2);
8     printf("Your input numbers: %d %d\n", number1, number2);
9
10    printf("Please input two numbers (use - for separation): ");
11    scanf("%d-%d", &number1, &number2);
12    printf("Your input numbers: %d-%d\n", number1, number2);
13
14    return 0;
15 }
```

```
Please input two numbers (use space for separation): 10 20
Your input numbers: 10 20
Please input two numbers (use - for separation): 10-20
Your input numbers: 10-20
```

Variables

- Variable are stored in the computer's memory.
- For instance

```
int integer1 = 45;  
int integer2 = 72;
```

Variables

- Variable are stored in the computer's memory.
- For instance

```
int integer1 = 45;  
int integer2 = 72;
```

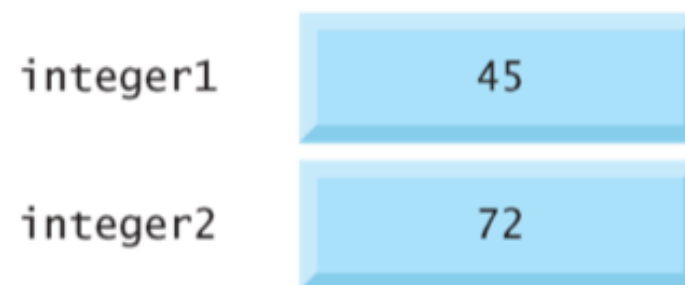


Fig. 2.7 | Memory locations after both variables are input.

Variables

- Naming variables
 - A series of characters consisting of letters, digits and underscores (_)
 - Does not begin with a digit!!
 - Clear and understandable, for example:

Variables

- Naming variables
 - A series of characters consisting of letters, digits and underscores (_)
 - Does not begin with a digit!!
 - Clear and understandable, for example:

```
int  ageOfStudent;  
int  ageOfTeacher;
```

Variables

- Naming variables
 - A series of characters consisting of letters, digits and underscores (_)
 - Does not begin with a digit!!
 - Clear and understandable, for example:

```
int ageOfStudent;  
int ageOfTeacher;
```

```
int age_of_student;  
int age_of_teacher;
```

Variables

- Constant number
 - Use “**const**” keyword to declare a constant number, for example

```
const float PI = 3.14;
```

- After declaring a constant number, we cannot change its value.

```
PI = 3.14159; // compiler error!!
```


Operators

Arithmetic Operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Remainder	%	$r \bmod s$	<code>r % s</code>

Fig. 2.9 | Arithmetic operators.

Type Conversion (Typecasting)

- In computer science, **type conversion**, **typecasting**, and coercion refers to different ways of, implicitly or explicitly, changing a variable of one data type into another.
- For example:

```
float aFloatNum = 3.2;  
int aIntNum = (int) aFloatNum;
```

Type Conversion (Typecasting)

- Example: [typecasting.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i_value = 3;
5     float f_value = 5.2;
6
7     printf("Before typecasting\n");
8     printf("The integer is: %i\n", i_value);
9     printf("The float is:  %f\n\n", f_value);
10
11     printf("After typecasting\n");
12     printf("The integer is: %f\n", (float) i_value);
13     printf("The float is:  %i\n\n", (int) f_value);
14
15     double da = 3.3;
16     double db = 3.3;
17     double dc = 3.4;
18     int result = (int)da + (int)db + (int)dc; //result == 9
19     printf("Results: %d\n", result);
20
21     return 0;
22 }
```

Relational and Conditional Operators

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	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.

Relational and Conditional Operators

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	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.

Relational and Conditional Operators

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	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.

```
10 > 5      1
10 >= 5     1
10 < 5      0
10 <= 5     0
10 == 5     0
10 != 5     1
```

Relational and Conditional Operators

- Conditional Operators

條件運算子	用法	傳回 true 的條件
&&	$X \ \&\& \ Y$	X 與 Y 必須同時為 true X, Y 不一定進行求值
	$X \ \ Y$	X 與 Y 之一為 true 即可 X, Y 不一定進行求值
!	$! X$	X 為 false

Relational and Conditional Operators

- For Conditional Operators
- The syntax of the conditional operator is

ex1 ? ex2 : ex3

- For instance

aIntNum = (n > 0) ? 10 : 0

Relational and Conditional Operators

- For Conditional Operators
- The syntax of the conditional operator is

ex1 ? ex2 : ex3

- For instance

aIntNum = (n > 0) ? 10 : 0

```
if (n > 0) {  
    aIntNum = 10;  
} else {  
    aIntNum = 0;  
}
```

Relational and Conditional Operators

- Example: [condoperator.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int score = 0;
5
6     printf("Please input a score: \n");
7     scanf("%d", &score);
8     printf("Pass or not: %c\n\n", (score >= 60 ? 'Y' : 'N'));
9
10    int input = 0;
11
12    printf("Please input an integer: \n");
13    scanf("%d", &input);
14    printf("Odd or not: %c\n", (input%2 ? 'Y' : 'N'));
15
16    return 0;
17 }
```

Relational and Conditional Operators

- Example: [condoperator.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int score = 0;
5
6     printf("Please input a score: \n");
7     scanf("%d", &score);
8     printf("Pass or not: %c\n\n", (score >= 60 ? 'Y' : 'N'));
9
10    int input = 0;
11
12    printf("Please input an integer: \n");
13    scanf("%d", &input);
14    printf("Odd or not: %c\n", (input%2 ? 'Y' : 'N'));
15
16    return 0;
17 }
```

Relational and Conditional Operators

- Example: [condoperator.c](#)

```
1 #include <stdio.h>
2
3 int main(void) {
4     int score = 0;
5
6     printf("Please input a score: \n");
7     scanf("%d", &score);
8     printf("Pass or not: %c\n\n", (score >= 60 ? 'Y' : 'N'));
9
10    int input = 0;
11
12    printf("Please input an integer: \n");
13    scanf("%d", &input);
14    printf("Odd or not: %c\n", (input%2 ? 'Y' : 'N'));
15
16    return 0;
17 }
```

Relational and Conditional Operators

- Example: `condoperator.c`

```
1 #include <stdio.h>
2
3 int main(void) {
4     int score = 0;
5
6     printf("Please input a score: \n");
7     scanf("%d", &score);
8     printf("Pass or not: %c\n\n", (score >= 60 ? 'Y' : 'N'));
9
10    int input = 0;
11
12    printf("Please input an integer: \n");
13    scanf("%d", &input);
14    printf("Odd or not: %c\n", (input%2 ? 'Y' : 'N'));
15
16    return 0;
17 }
```

```
Please input a score:
55
Pass or not: N

Please input an integer:
6
Odd or not: N
```

Logical and Bitwise Operators

位元運算子	用法	功能
&	$X \& Y$	X 與 Y 以 bit 為單位，做 AND 運算
	$X Y$	X 與 Y 以 bit 為單位，做 OR 運算
~	$\sim X$	X 做補數運算
^	$X \wedge Y$	X 與 Y 以 bit 為單位，做 XOR 運算
>>	$X \gg Y$	將 X 向右移動 Y 個位元
<<	$X \ll Y$	將 X 向左移動 Y 個位元

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \underline{\vee} Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \sqcup Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \underline{\vee} Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \underline{\vee} Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \sqcup Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Truth Table

P	Q	$P \wedge Q$	$P \vee Q$	$P \triangle Q$	$P \underline{\vee} Q$	$P \rightarrow Q$	$P \leftarrow Q$
F	F	F	F	F	T	T	T
F	T	F	T	T	F	T	F
T	F	F	T	T	F	F	T
T	T	T	T	F	T	T	T

Logical and Bitwise Operators

- Example: [bitoperation.c](#)

```
5  printf("AND operations: ");
6  printf("0 AND 0\t\t%d\n", 0 & 0);
7  printf("0 AND 1\t\t%d\n", 0 & 1);
8  printf("1 AND 0\t\t%d\n", 1 & 0);
9  printf("1 AND 1\t\t%d\n\n", 1 & 1);
10
11 printf("OR operations: ");
12 printf("0 OR 0\t\t%d\n", 0 | 0);
13 printf("0 OR 1\t\t%d\n", 0 | 1);
14 printf("1 OR 0\t\t%d\n", 1 | 0);
15 printf("1 OR 1\t\t%d\n\n", 1 | 1);
16
17 printf("XOR operations: ");
18 printf("0 XOR 0\t\t%d\n", 0 ^ 0);
19 printf("0 XOR 1\t\t%d\n", 0 ^ 1);
20 printf("1 XOR 0\t\t%d\n", 1 ^ 0);
21 printf("1 XOR 1\t\t%d\n\n", 1 ^ 1);
```

Logical and Bitwise Operators

- Example: [bitoperation.c](#)

```
5  printf("AND operations: ");
6  printf("0 AND 0\t\t%d\n", 0 & 0);
7  printf("0 AND 1\t\t%d\n", 0 & 1);
8  printf("1 AND 0\t\t%d\n", 1 & 0);
9  printf("1 AND 1\t\t%d\n\n", 1 & 1);
10
11 printf("OR operations: ");
12 printf("0 OR 0\t\t%d\n", 0 | 0);
13 printf("0 OR 1\t\t%d\n", 0 | 1);
14 printf("1 OR 0\t\t%d\n", 1 | 0);
15 printf("1 OR 1\t\t%d\n\n", 1 | 1);
16
17 printf("XOR operations: ");
18 printf("0 XOR 0\t\t%d\n", 0 ^ 0);
19 printf("0 XOR 1\t\t%d\n", 0 ^ 1);
20 printf("1 XOR 0\t\t%d\n", 1 ^ 0);
21 printf("1 XOR 1\t\t%d\n\n", 1 ^ 1);
```

```
29  printf("Shift operations: \n");
30
31  int num = 1;
32
33  printf("2^0: %d\n", num);
34
35  num = num << 1;
36  printf("2^1: %d\n", num);
37
38  num = num << 1;
39  printf("2^2: %d\n", num);
40
41  num = num << 1;
42  printf("2^3: %d\n", num);
```

Logical and Bitwise Operators

- Example: [bitoperation.c](#)

```
5  printf("AND operations: ");
6  printf("0 AND 0\t\t%d\n", 0 & 0);
7  printf("0 AND 1\t\t%d\n", 0 & 1);
8  printf("1 AND 0\t\t%d\n", 1 & 0);
9  printf("1 AND 1\t\t%d\n\n", 1 & 1);
10
11 printf("OR operations: ");
12 printf("0 OR 0\t\t%d\n", 0 | 0);
13 printf("0 OR 1\t\t%d\n", 0 | 1);
14 printf("1 OR 0\t\t%d\n", 1 | 0);
15 printf("1 OR 1\t\t%d\n\n", 1 | 1);
16
17 printf("XOR operations: ");
18 printf("0 XOR 0\t\t%d\n", 0 ^ 0);
19 printf("0 XOR 1\t\t%d\n", 0 ^ 1);
20 printf("1 XOR 0\t\t%d\n", 1 ^ 0);
21 printf("1 XOR 1\t\t%d\n\n", 1 ^ 1);
```

```
29  printf("Shift operations: \n");
30
31  int num = 1;
32
33  printf("2^0: %d\n", num);
34
35  num = num << 1;
36  printf("2^1: %d\n", num);
37
38  num = num << 1;
39  printf("2^2: %d\n", num);
40
41  num = num << 1;
42  printf("2^3: %d\n", num);
```


Logical and Bitwise Operators

- Example: [bitoperation.c](#)

```
5  printf("AND operations: ");
6  printf("0 AND 0\t\t%d\n", 0 & 0);
7  printf("0 AND 1\t\t%d\n", 0 & 1);
8  printf("1 AND 0\t\t%d\n", 1 & 0);
9  printf("1 AND 1\t\t%d\n\n", 1 & 1);
10
11 printf("OR operations: ");
12 printf("0 OR 0\t\t%d\n", 0 | 0);
13 printf("0 OR 1\t\t%d\n", 0 | 1);
14 printf("1 OR 0\t\t%d\n", 1 | 0);
15 printf("1 OR 1\t\t%d\n\n", 1 | 1);
16
17 printf("XOR operations: ");
18 printf("0 XOR 0\t\t%d\n", 0 ^ 0);
19 printf("0 XOR 1\t\t%d\n", 0 ^ 1);
20 printf("1 XOR 0\t\t%d\n", 1 ^ 0);
21 printf("1 XOR 1\t\t%d\n\n", 1 ^ 1);
```

```
29  printf("Shift operations: \n");
30
31  int num = 1;
32
33  printf("2^0: %d\n", num);
34
35  num = num << 1;
36  printf("2^1: %d\n", num);
37
38  num = num << 1;
39  printf("2^2: %d\n", num);
40
41  num = num << 1;
42  printf("2^3: %d\n", num);
```

Logical and Bitwise Operators

- Example: [bitoperation.c](#)

```
5  printf("AND operations: ");
6  printf("0 AND 0\t\t%d\n", 0 & 0);
7  printf("0 AND 1\t\t%d\n", 0 & 1);
8  printf("1 AND 0\t\t%d\n", 1 & 0);
9  printf("1 AND 1\t\t%d\n\n", 1 & 1);
10
11 printf("OR operations: ");
12 printf("0 OR 0\t\t%d\n", 0 | 0);
13 printf("0 OR 1\t\t%d\n", 0 | 1);
14 printf("1 OR 0\t\t%d\n", 1 | 0);
15 printf("1 OR 1\t\t%d\n\n", 1 | 1);
16
17 printf("XOR operations: ");
18 printf("0 XOR 0\t\t%d\n", 0 ^ 0);
19 printf("0 XOR 1\t\t%d\n", 0 ^ 1);
20 printf("1 XOR 0\t\t%d\n", 1 ^ 0);
21 printf("1 XOR 1\t\t%d\n\n", 1 ^ 1);
```

```
29  printf("Shift operations: \n");
30
31  int num = 1;
32
33  printf("2^0: %d\n", num);
34
35  num = num << 1;
36  printf("2^1: %d\n", num);
37
38  num = num << 1;
39  printf("2^2: %d\n", num);
40
41  num = num << 1;
42  printf("2^3: %d\n", num);
```

```
AND operations: 0 AND 0      0
0 AND 1      0
1 AND 0      0
1 AND 1      1

OR operations: 0 OR 0      0
0 OR 1      1
1 OR 0      1
1 OR 1      1

XOR operations: 0 XOR 0      0
0 XOR 1      1
1 XOR 0      1
1 XOR 1      0

NOT operations: NOT 0      1
NOT 1      0

=====
Shift operations:
2^0: 1
2^1: 2
2^2: 4
2^3: 8
```

Increment, Decrement, and Assignment operations

- Assignment operations
 - Goal: store a value into a variable

指定運算子	用法	相當於
=	$X = Y$	$X = Y$
+=	$X += Y$	$X = X + Y$
-=	$X -= Y$	$X = X - Y$
*=	$X *= Y$	$X = X * Y$
/=	$X /= Y$	$X = X / Y$
%=	$X \% = Y$	$X = X \% Y$

Increment, Decrement, and Assignment operations

- Assignment operations
 - Goal: store a value into a variable

指定運算子	用法	相當於
=	$X = Y$	$X = Y$
+=	$X += Y$	$X = X + Y$
-=	$X -= Y$	$X = X - Y$
*=	$X *= Y$	$X = X * Y$
/=	$X /= Y$	$X = X / Y$
%=	$X \% = Y$	$X = X \% Y$

Increment, Decrement, and Assignment operations

增減運算子	用法	說明
++ (前置式)	++X	$X = X + 1$
-- (前置式)	-- X	$X = X - 1$
(後置式) ++	X++	$X = X + 1$
(後置式) --	X--	$X = X - 1$

```
int x = 3;  
printf("%d\n", ++x);  
printf("%d\n", x)
```

```
int x = 3;  
printf("%d\n", x++);  
printf("%d\n", x)
```

Increment, Decrement, and Assignment operations

- Example: [prefix_postfix.c](#)

```
5  int x = 3;
6
7  printf("Prefix increment operation:");
8  printf("%d\n", ++x);
9  printf("%d\n", x);
10
11 x = 3;
12
13 printf("Postfix increment operation:");
14 printf("%d\n", x++);
15 printf("%d\n", x);
```

```
Prefix increment operation:
4
4

Postfix increment operation:
3
4
```

C Structured Program Development

Chapter 3

- 3.1 Introduction
- 3.2 Algorithms
- 3.3 Pseudocode
- 3.4 Control Structures
- 3.5 The `if` Selection Statement
- 3.6 The `if...else` Selection Statement
- 3.7 The `while` Repetition Statement
- 3.8 Formulating Algorithms Case Study 1: Counter-Controlled Repetition
Counter-Controlled Repetition
- 3.9 Formulating Algorithms with Top-Down, Stepwise Refinement Case Study
2: Sentinel-Controlled Repetition Sentinel-Controlled Repetition
- 3.10 Formulating Algorithms with Top-Down, Stepwise Refinement Case Study
3: Nested Control Structures Nested Control Structures
- 3.11 Assignment Operators
- 3.12 Increment and Decrement Operators

Introduction

- Before writing a program to solve a particular problem, it's essential to have a thorough understanding of the problem and a carefully planned approach to solving the problem.

Algorithms

- A procedure for solving a problem in terms of
 - the actions to be executed, and
 - the order in which these actions are to be executed

Pseudocode

- Pseudocode is an artificial and informal language that helps you develop algorithms.
- For instance

```
Set moveCount to 1
FOR each row on the board
  FOR each column on the board
    IF gameBoard position (row, column) is occupied THEN
      CALL findAdjacentTiles with row, column
      INCREMENT moveCount
    END IF
  END FOR
END FOR
```

Control Structures

- Sequential execution
 - Statements in a program are executed one after the other in the order in which they're written.
- Three control structures
 - Sequence structure
 - Selection structure
 - Repetition structure

Sequence structure

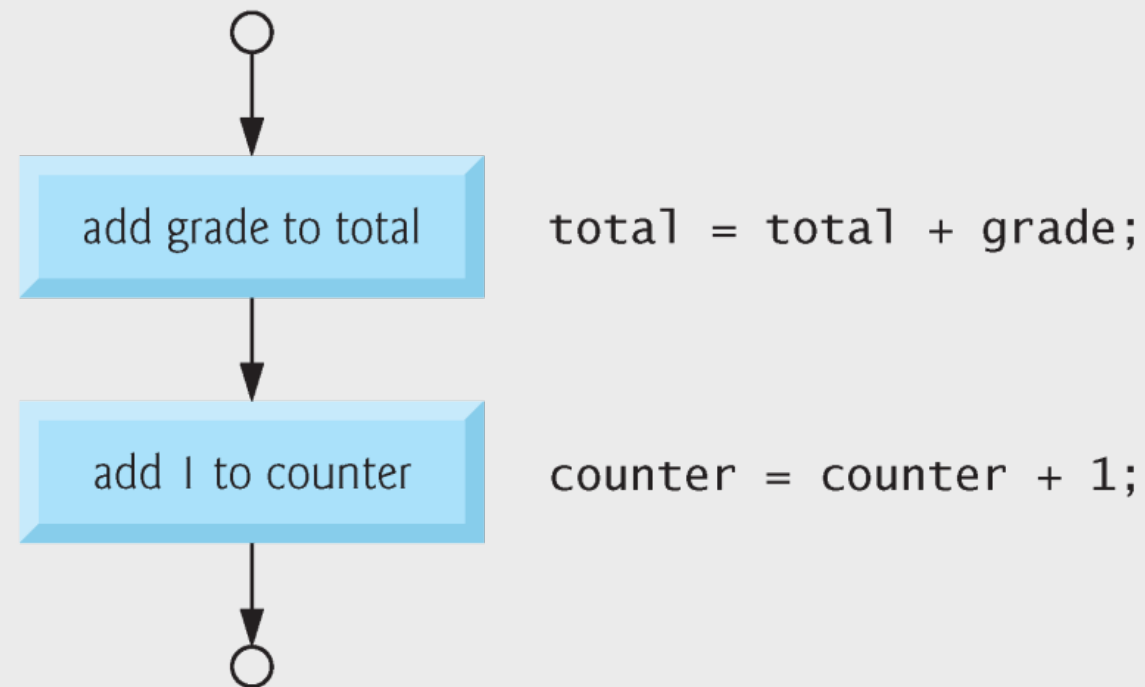


Fig. 3.1 | Flowcharting C's sequence structure.

Selection Statement

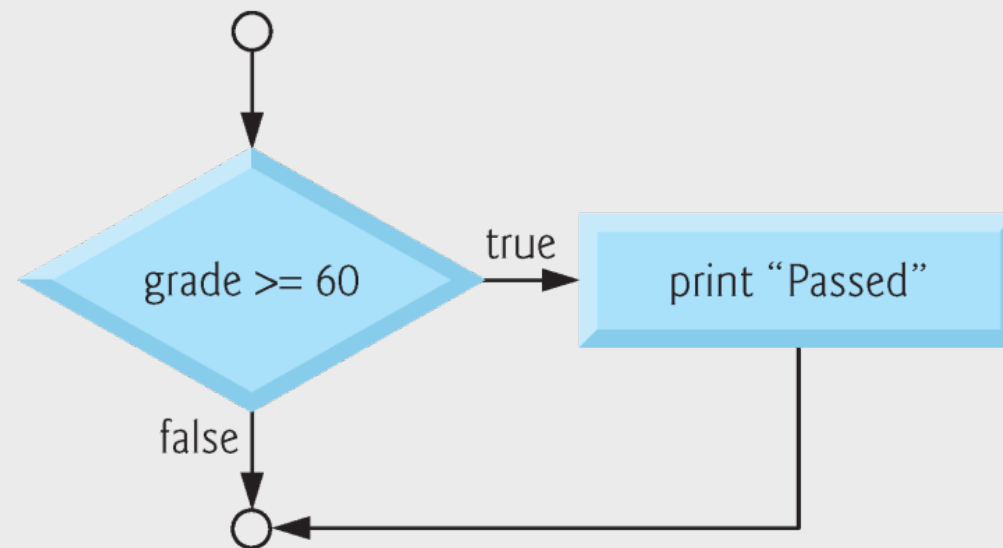


Fig. 3.2 | Flowcharting the single-selection `if` statement.

Repetition structure

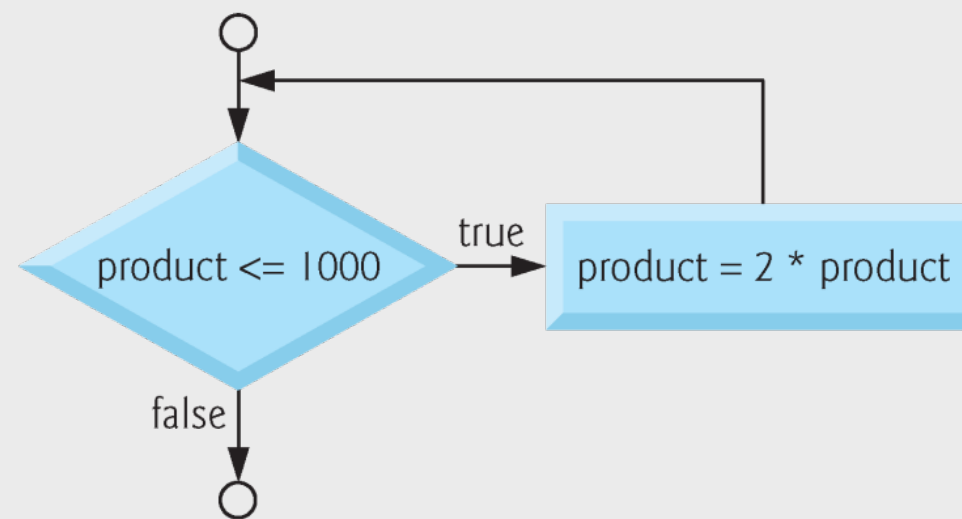


Fig. 3.4 | Flowcharting the while repetition statement.

Selection Statement

The **if** Selection Statement

- The preceding pseudocode If statement may be written in C as

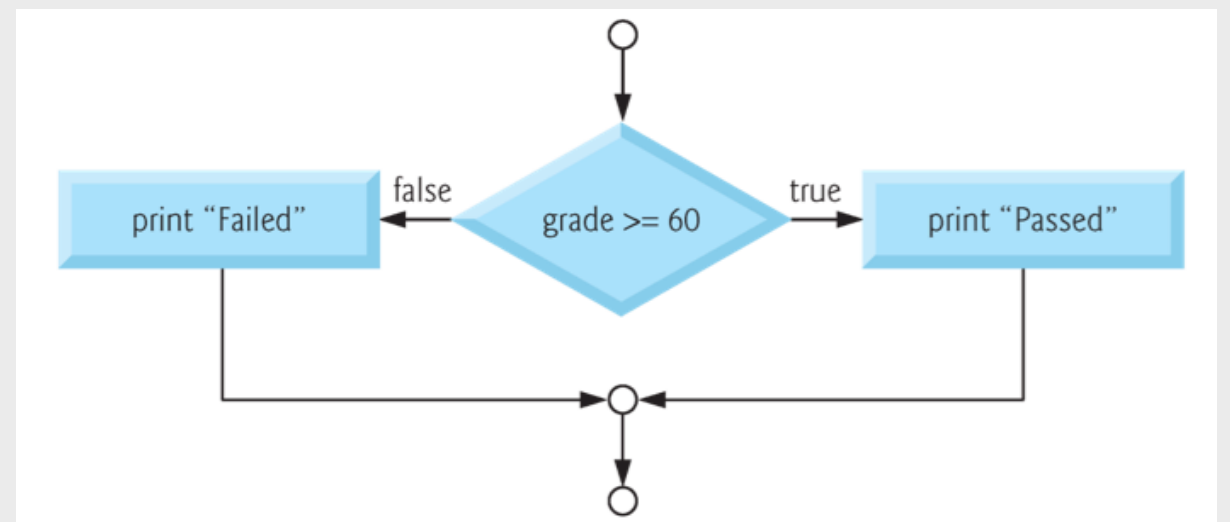
```
if ( grade >= 60 ) {  
    printf( "Passed\n" );  
} /* end if */
```

- Decisions can be based on conditions containing relational or equality operators.

The **if...else** Selection Statement

- The preceding pseudocode **if...else** statement may be written in C as

```
if ( grade >= 60 ) {  
    printf( "Passed\n" );  
} /* end if */  
else {  
    printf( "Failed\n" );  
} /* end else */
```



The **if...else** Selection Statement (Cont.)

- C provides the conditional operator (**? :**) which is closely related to the **if...else** statement
- For example, the printf statement

```
printf( "%s\n", grade >= 60 ?  
"Passed" : "Failed" );
```

The **if...else** Selection Statement (Cont.)

- Nested **if...else** statements test for multiple cases by placing **if...else** statements inside **if...else** statements.

The **if...else** Selection Statement (Cont.)

- Nested **if...else** statements test for multiple cases by placing **if...else** statements inside **if...else** statements.

```
if ( grade >= 90 )  
    printf( "A\n" );  
else if ( grade >= 80 )  
    printf( "B\n" );  
else if ( grade >= 70 )  
    printf( "C\n" );  
else if ( grade >= 60 )  
    printf( "D\n" );  
else  
    printf( "F\n" );
```

The **if...else** Selection Statement (Cont.)

- Nested **if...else** statements test for multiple cases by placing **if...else** statements inside **if...else** statements.

```
if ( grade >= 90 )  
    printf( "A\n" );  
else if ( grade >= 80 )  
    printf( "B\n" );  
else if ( grade >= 70 )  
    printf( "C\n" );  
else if ( grade >= 60 )  
    printf( "D\n" );  
else  
    printf( "F\n" );
```

The **if...else** Selection Statement (Cont.)

- The following example includes a compound statement in the **else** part of an **if...else** statement.

```
if ( grade >= 60 ) {  
    printf( "Passed.\n" );  
} /* end if */  
else {  
    printf( "Failed.\n" );  
    printf( "You must take this course again.\n" );  
} /* end else */
```

Repetition Statement

The **while** Repetition Statement

- A **repetition** statement allows you to specify that an action is to be **repeated** while some condition remains true.
- The statement(s) contained in the while repetition statement constitute the body of the while.
- The while statement body may be a **single statement** or a **compound statement**.

The **while** Repetition Statement (Cont.)

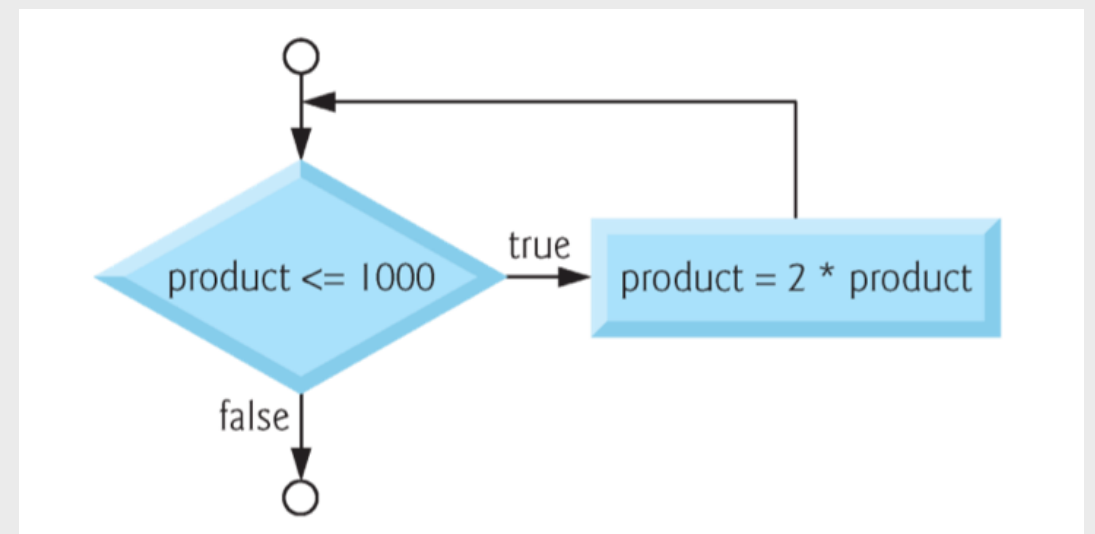
- As an example of an actual while, consider a program segment designed to find the first power of 2 larger than 1000.

```
product = 2;  
while ( product <= 1000 ) {  
    product = 2 * product;  
} /* end while */
```

The **while** Repetition Statement (Cont.)

- As an example of an actual while, consider a program segment designed to find the first power of 2 larger than 1000.

```
product = 2;  
while ( product <= 1000 ) {  
    product = 2 * product;  
} /* end while */
```



Counter-Controlled Repetition

- We use counter-controlled repetition to input the grades one at a time.
- This technique uses a variable called a counter to specify the number of times a set of statements should execute.

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```


Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```


Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

Example: fig03_06.c

```
8  int counter; /* number of grade to be entered next */
9  int grade; /* grade value */
10 int total; /* sum of grades input by user */
11 int average; /* average of grades */
12
13 /* initialization phase */
14 total = 0; /* initialize total */
15 counter = 1; /* initialize loop counter */
16
17 /* processing phase */
18 while ( counter <= 10 ) { /* loop 10 times */
19     printf( "Enter grade: " ); /* prompt for input */
20     scanf( "%d", &grade ); /* read grade from user */
21     total = total + grade; /* add grade to total */
22     counter = counter + 1; /* increment counter */
23 } /* end while */
24
25 /* termination phase */
26 average = total / 10; /* integer division */
27
28 printf( "Class average is %d\n", average ); /* display result */
29 return 0; /* indicate program ended successfully */
30 } /* end function main */
```

```
Enter grade: 10
Enter grade: 20
Enter grade: 30
Enter grade: 40
Enter grade: 50
Enter grade: 60
Enter grade: 70
Enter grade: 80
Enter grade: 90
Enter grade: 100
Class average is 55
```


Sentinel-Controlled Repetition

- One way to solve this problem is to use a special value called a **sentinel value** (also called a **flag value**) to indicate “end of data entry.”
- Sentinel-controlled repetition is often called **indefinite repetition** because the number of repetitions is not known before the loop begins executing.

Example: fig03_08.c

```
16  /* processing phase */
17  /* get first grade from user */
18  printf( "Enter grade, -1 to end: " ); /* prompt for input */
19  scanf( "%d", &grade ); /* read grade from user */
20
21  /* loop while sentinel value not yet read from user */
22  while ( grade != -1 ) {
23      total = total + grade; /* add grade to total */
24      counter = counter + 1; /* increment counter */
25
26      /* get next grade from user */
27      printf( "Enter grade, -1 to end: " ); /* prompt for input */
28      scanf( "%d", &grade ); /* read next grade */
29  } /* end while */
30
31  /* termination phase */
32  /* if user entered at least one grade */
33  if ( counter != 0 ) {
34      /* calculate average of all grades entered */
35      average = ( float ) total / counter; /* avoid truncation */
36      /* display average with two digits of precision */
37      printf( "Class average is %.2f\n", average ); ...
38  } else { /* if no grades were entered, output message */
39      printf( "No grades were entered\n" );
40  } /* end else */
```

Example: fig03_08.c

```
16  /* processing phase */
17  /* get first grade from user */
18  printf( "Enter grade, -1 to end: " ); /* prompt for input */
19  scanf( "%d", &grade ); /* read grade from user */
20
21  /* loop while sentinel value not yet read from user */
22  while ( grade != -1 ) {
23      total = total + grade; /* add grade to total */
24      counter = counter + 1; /* increment counter */
25
26      /* get next grade from user */
27      printf( "Enter grade, -1 to end: " ); /* prompt for input */
28      scanf( "%d", &grade ); /* read next grade */
29  } /* end while */
30
31  /* termination phase */
32  /* if user entered at least one grade */
33  if ( counter != 0 ) {
34      /* calculate average of all grades entered */
35      average = ( float ) total / counter; /* avoid truncation */
36      /* display average with two digits of precision */
37      printf( "Class average is %.2f\n", average ); ...
38  } else { /* if no grades were entered, output message */
39      printf( "No grades were entered\n" );
40  } /* end else */
```

Example: fig03_08.c

```
16  /* processing phase */
17  /* get first grade from user */
18  printf( "Enter grade, -1 to end: " ); /* prompt for input */
19  scanf( "%d", &grade ); /* read grade from user */
20
21  /* loop while sentinel value not yet read from user */
22  while ( grade != -1 ) {
23      total = total + grade; /* add grade to total */
24      counter = counter + 1; /* increment counter */
25
26      /* get next grade from user */
27      printf( "Enter grade, -1 to end: " ); /* prompt for input */
28      scanf( "%d", &grade ); /* read next grade */
29  } /* end while */
30
31  /* termination phase */
32  /* if user entered at least one grade */
33  if ( counter != 0 ) {
34      /* calculate average of all grades entered */
35      average = ( float ) total / counter; /* avoid truncation */
36      /* display average with two digits of precision */
37      printf( "Class average is %.2f\n", average ); ...
38  } else { /* if no grades were entered, output message */
39      printf( "No grades were entered\n" );
40  } /* end else */
```

```
Enter grade, -1 to end: 10
Enter grade, -1 to end: 20
Enter grade, -1 to end: 30
Enter grade, -1 to end: -1
Class average is 20.00
```

Example: fig03_08.c

```
16  /* processing phase */
17  /* get first grade from user */
18  printf( "Enter grade, -1 to end: " ); /* prompt for input */
19  scanf( "%d", &grade ); /* read grade from user */
20
21  /* loop while sentinel value not yet read from user */
22  while ( grade != -1 ) {
23      total = total + grade; /* add grade to total */
24      counter = counter + 1; /* increment counter */
25
26      /* get next grade from user */
27      printf( "Enter grade, -1 to end: " ); /* prompt for input */
28      scanf( "%d", &grade ); /* read next grade */
29  } /* end while */
30
31  /* termination phase */
32  /* if user entered at least one grade */
33  if ( counter != 0 ) {
34      /* calculate average of all grades entered */
35      average = ( float ) total / counter; /* avoid truncation */
36      /* display average with two digits of precision */
37      printf( "Class average is %.2f\n", average ); ...
38  } else { /* if no grades were entered, output message */
39      printf( "No grades were entered\n" );
40  } /* end else */
```

```
Enter grade, -1 to end: 10
Enter grade, -1 to end: 20
Enter grade, -1 to end: 30
Enter grade, -1 to end: -1
Class average is 20.00
```

```
Enter grade, -1 to end: -1
No grades were entered
```

Example: `fig03_08.c`

- Line 35

```
average = ( float ) total / counter;
```

- explicit conversion
- If the `%f` conversion specifier is used (without specifying the precision), **the default precision of 6** is used—exactly as if the conversion specifier `%.2f` had been used.

Nested Control Structures

- In this case study we'll see the only other structured way control statements may be connected in C, namely through **nesting** of one control statement within another.

Example: fig03_10.c

```
14  /* process 10 students using counter-controlled loop */
15  while ( student <= 10 ) {
16
17      /* prompt user for input and obtain value from user */
18      printf( "Enter result ( 1=pass,2=fail ): " );
19      scanf( "%d", &result );
20
21      /* if result 1, increment passes */
22      if ( result == 1 ) {
23          passes = passes + 1;
24      } /* end if */
25      else { /* otherwise, increment failures */
26          failures = failures + 1;
27      } /* end else */
28
29      student = student + 1; /* increment student counter */
30  } /* end while */
31
32  /* termination phase; display number of passes and failures */
33  printf( "Passed %d\n", passes );
34  printf( "Failed %d\n", failures );
35
36  /* if more than eight students passed, print "Bonus to instructor!" */
37  if ( passes > 8 ) {
38      printf( "Bonus to instructor!\n" );
39  } /* end if */
```


Example: fig03_10.c

```
14  /* process 10 students using counter-controlled loop */
15  while ( student <= 10 ) {
16
17      /* prompt user for input and obtain value from user */
18      printf( "Enter result ( 1=pass,2=fail ): " );
19      scanf( "%d", &result );
20
21      /* if result 1, increment passes */
22      if ( result == 1 ) {
23          passes = passes + 1;
24      } /* end if */
25      else { /* otherwise, increment failures */
26          failures = failures + 1;
27      } /* end else */
28
29      student = student + 1; /* increment student counter */.
30  } /* end while */
31
32  /* termination phase; display number of passes and failures */
33  printf( "Passed %d\n", passes );
34  printf( "Failed %d\n", failures );
35
36  /* if more than eight students passed, print "Bonus to instructor!" */
37  if ( passes > 8 ) {
38      printf( "Bonus to instructor!\n" );
39  } /* end if */
```

```
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Passed 5
Failed 5
```

Example: fig03_10.c

```
14  /* process 10 students using counter-controlled loop */
15  while ( student <= 10 ) {
16
17      /* prompt user for input and obtain value from user */
18      printf( "Enter result ( 1=pass,2=fail ): " );
19      scanf( "%d", &result );
20
21      /* if result 1, increment passes */
22      if ( result == 1 ) {
23          passes = passes + 1;
24      } /* end if */
25      else { /* otherwise, increment failures */
26          failures = failures + 1;
27      } /* end else */
28
29      student = student + 1; /* increment student counter */.
30  } /* end while */
31
32  /* termination phase; display number of passes and failures */
33  printf( "Passed %d\n", passes );
34  printf( "Failed %d\n", failures );
35
36  /* if more than eight students passed, print "Bonus to instructor!" */
37  if ( passes > 8 ) {
38      printf( "Bonus to instructor!\n" );
39  } /* end if */
```

```
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Passed 5
Failed 5
```

```
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 2
Enter result ( 1=pass,2=fail ): 1
Enter result ( 1=pass,2=fail ): 1
Passed 9
Failed 1
Bonus to instructor!
```