w03-Lab

Variables, Data Types, Expression, and Assignment Part II

Assembled for 204111

by Areerat Trongratsameethong

Topics

- **Assignment**
- Type Conversions
- Accumulating
- Operator Precedence
- Increment and Decrement Operators
- printf() Function
- scanf() Function
- Common Programming Errors

Assignment

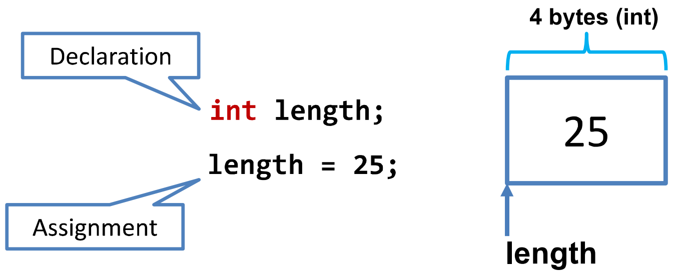
- **The general syntax for an assignment statement is**

```
variable = operand;
```

  - **The operand to the right of the assignment operator (=) can be a constant, a variable, or an expression**
- **The equal sign in C does not have the same meaning as an equal sign in algebra**

Assignment (2)

- **length = 25; is read “length is assigned the value 25” or “length ← 25”**



```
int length;
length = 25;
```

Assignment (3)

- **Subsequent assignment statements can be used to change the value assigned to a variable**

```
length = 3.7;
length = 6.28; // เราสามารถกำหนดค่าข้อมูลใหม่ให้กับตัวแปรได้
```

- **The <u>operand</u> to the right of the equal sign in an assignment statement <u>can be a variable or any valid C expression</u>**

```
sum = 3 + 7;
product = .05 * 14.6;
```

Assignment with Expression

```
sum = 3 + 7;
```

Calculate 3+7

Store 10 to sum

```
product = .05 * 14.6;
```

Calculate .05 *14.6

Store 0.73 to product

The value of the expression to the <u>right</u> of = is <u>computed first</u> and <u>then</u> the calculated value is <u>stored</u> in the variable to the <u>left</u> of =

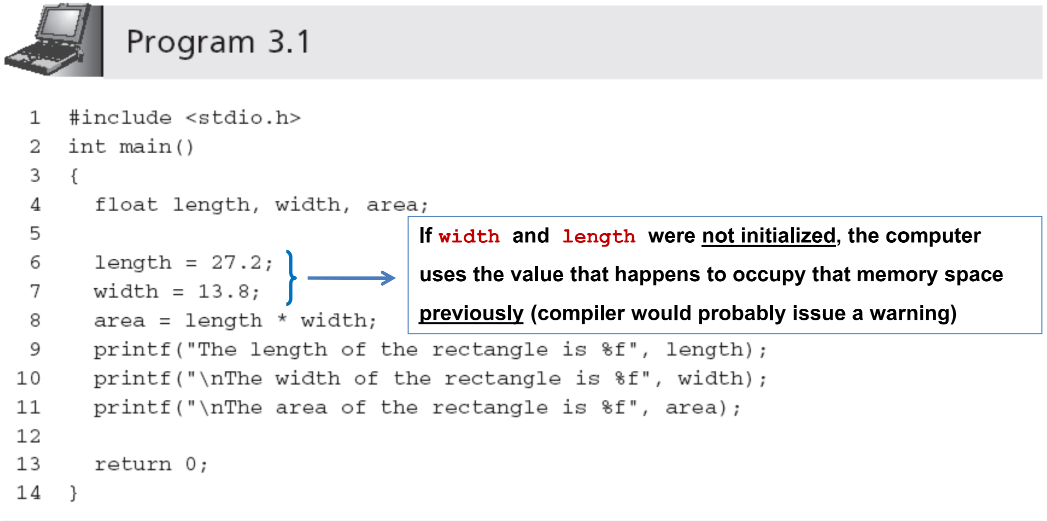
Assignment (4)

- **Variables used in the expression to the right of the = must be initialized if the result is to make sense**

```
int x;
int y = 20 * x;
```

- **INVALID Assignment**
  - `amount + 1892 = 1000 + 10 * 5;`

Assignment (5)



Program 3.1

```
#include <stdio.h>
int main()
{
  float length, width, area;

  length = 27.2;
  width = 13.8;
  area = length * width;
  printf("The length of the rectangle is %f", length);
  printf("\nThe width of the rectangle is %f", width);
  printf("\nThe area of the rectangle is %f", area);

  return 0;
}
```

If `width` and `length` were <u>not initialized</u>, the computer uses the value that happens to occupy that memory space <u>previously</u> (compiler would probably issue a warning)

Assignment (6)

```
1  #include <stdio.h>
2
3  int  main()
4  {
5    int sum;
6    sum = 25;
7    printf(“\nThe number stored in sum is %d.”, sum);
8    sum = sum + 10;
9    printf(“ \nThe number now stored in sum is %d.”, sum);
10   return 0;
11 }
```

Two major steps for: sum = sum + 10;

Line 6: sum = ?

Line 8: sum = ?

Step 1:

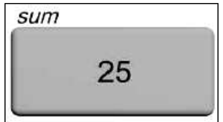


Figure 3.1 The integer 25 is stored in sum

Step 2:

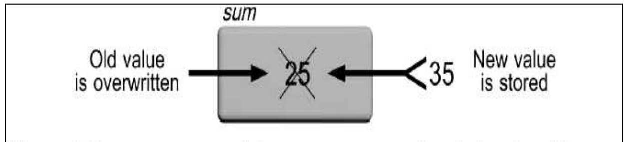


Figure 3.2 sum = sum + 10; causes a new value to be stored in sum

Assignment Variations

- **Assignment expressions like**

  `sum = sum + 25`

  can be written using the following operators:

  += -= *= /= %=

Example:

```
sum = sum + 10;          // is equivalent to below statement
sum += 10;

price = price * rate;    // is equivalent to below statement
price *= rate;

price *= rate + 1;       // is equivalent to below statement
price = price * (rate + 1);
```

Topics

- Assignment
- **Type Conversions**
- Accumulating
- Operator Precedence
- Increment and Decrement Operators
- printf() Function
- scanf() Function
- Common Programming Errors

Implicit Type Conversions

- **Data type conversions take place across assignment operators**

  ```
  double result;
  result = 4;  // integer literal 4 is converted to 4.0
  ```

- **The automatic conversion across an assignment operator is called an implicit type conversion (implicit casting)**

  ```
  int answer;
  answer = 2.764;  // double literal 2.764 is converted to 2
  ```

  - **Here the implicit conversion is from a higher precision to a lower precision data type; the compiler will issue a warning**

Explicit Type Conversions

- The operator used to force the type conversion of a value to another type is the cast operator with the form

  `(dataType) expression`

  - where dataType is the desired data type of the expression following the cast
  - This is called explicit type conversion (explicit casting)

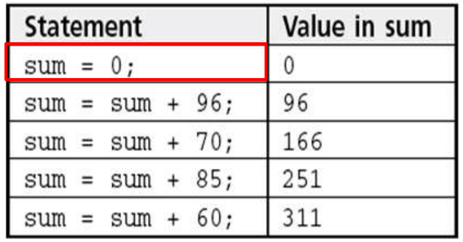
```
double sum;
printf("converting to int %d",((int) sum));
```

- If sum is declared as `double sum;`
  - `(int) sum` is the integer value determined by truncating sum’s fractional part

Topics

- Assignment
- Type Conversions
- **Accumulating**
- Operator Precedence
- Increment and Decrement Operators
- printf() Function
- scanf() Function
- Common Programming Errors

Accumulating



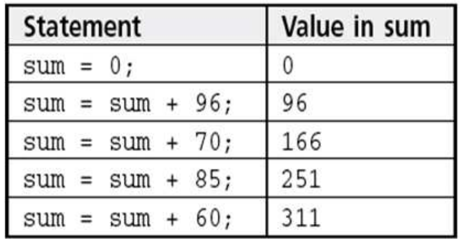
Statement	Value in sum
sum = 0;	0
sum = sum + 96;	96
sum = sum + 70;	166
sum = sum + 85;	251
sum = sum + 60;	311

- Accumulating: การเพิ่มค่าข้อมูลของ variables ไปเรื่อย ๆ เช่น ในกรณีต้องการหาค่าผลรวม (sum) เพื่อนำไปหาค่าเฉลี่ย

A previously stored number, if it has not been initialized to a specific and known value, is frequently called a garbage value

- The first statement initializes sum to 0
  - This removes any previously stored value in sum that would invalidate the final total

Accumulating (2)



Statement	Value in sum
sum = 0;	0
sum = sum + 96;	96
sum = sum + 70;	166
sum = sum + 85;	251
sum = sum + 60;	311

```
#include <stdio.h>

int main() {

    int sum;
    sum = 0;
    printf("\nThe value of sum is initially set to %d.", sum);
    sum = sum + 96;
    printf("\n sum is now %d.", sum);
    sum = sum + 70;
    printf("\n sum is now %d.", sum);
    sum = sum + 85;
    printf("\n sum is now %d.", sum);
    sum = sum + 60;
    printf("\n The final sum is %d.", sum);

    return 0;
}
```

Topics

- Assignment
- Type Conversions
- Accumulating
- **Operator Precedence**
- Increment and Decrement Operators
- printf() Function
- scanf() Function
- Common Programming Errors

Reference: http://www.difranco.net/compsci/C_Operator_Precedence_Table.htm

Operator Precedence

high

Precedence	Operator	Description	Associativity
1	() ++ --	Parentheses Suffix/postfix increment and decrement	Left-to-right
2	++ -- + − *(type)*	Prefix increment and decrement Unary plus and minus Type cast	Right-to-left
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ −	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
15	,	Comma	Left-to-right

low

Arithmetic

Logical

Bitwise

Logical

Arithmetic Operator Precedence

- Two binary arithmetic operator symbols must never be placed side by side

```
2 + + 5; // invalid
  5+ -2; // valid
```

- Parentheses may be used to form groupings
  - Expressions in parentheses are evaluated first

```
(4 + 5) * 3 % 2; // The (4 +5) is performed first
```

- Parentheses may be enclosed by other parentheses

```
2 % ((4 + 5) * 3)); // The (4 + 5) is performed first
                    // The 9 * 3 is performed second
```

- Parentheses cannot be used to indicate multiplication

```
2 % (4 + 5)(3 - 2); // invalid
```

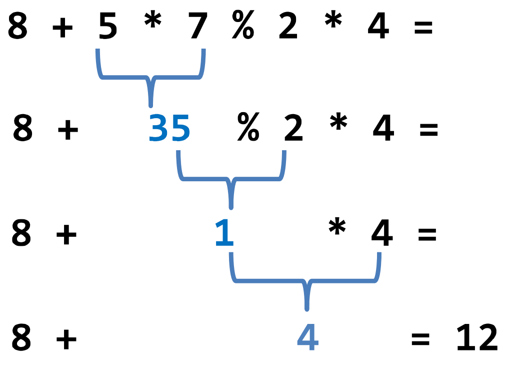
Arithmetic Operator Precedence (2)

- Three basic levels of precedence:
  - All negations are done first
  - Multiplication, division, and modulo operations are computed next; expressions containing more than one of these operators are evaluated from left to right as each operator is encountered
  - Addition and subtraction are computed last; expressions containing more than one addition or subtraction are evaluated from left to right as each operator is encountered

Arithmetic Operator Precedence (3)

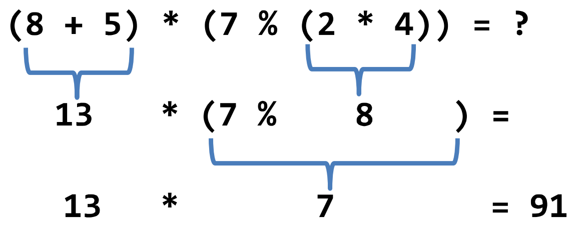
`(8 + 5) * (7 % (2 * 4)) = ?`

- Example:



```
8 + 5 * 7 % 2 * 4 =
8 +  35  % 2 * 4 =
8 +      1   * 4 =
8 +          4     = 12
```

Arithmetic Operator Precedence (4)



```
(8 + 5) * (7 % (2 * 4)) = ?
  13    * (7 %    8    ) =
  13    *      7         = 91
```

Assignment Operator

Multiple Assignments

- = has the lowest precedence of all the binary and unary arithmetic operators
- Multiple assignments are possible in the same statement

  a = b = c = 25;

  - All = operators have the same precedence
  - Operator has right-to-left associativity
    - c = 25;// step 1
    - b = c; // step 2
    - a = b; // step 3

Topics

- Assignment
- Type Conversions
- Accumulating
- Operator Precedence
- **Increment and Decrement Operators**
- printf() Function
- scanf() Function
- Common Programming Errors

Increment and Decrement Operators

- A counting statement is very similar to the accumulating statement

  variable = *variable* + *fixedNumber*;

Examples:

```
i = i + 1;
m = m + 2;
```

- Increment operator (++):

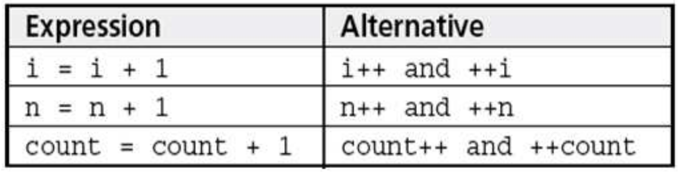
  variable = *variable* + 1;

  - can be replaced by

```
variable++; //or
++variable;
```

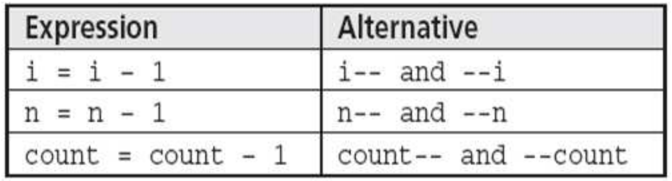
Increment and Decrement Operators (2)

Table 3.2 Examples of the Increment Operator



Expression	Alternative
`i = i + 1`	`i++` and `++i`
`n = n + 1`	`n++` and `++n`
`count = count + 1`	`count++` and `++count`

Table 3.3 Examples of the Decrement Operator



Expression	Alternative
`i = i - 1`	`i--` and `--i`
`n = n - 1`	`n--` and `--n`
`count = count - 1`	`count--` and `--count`

Increment and Decrement Operators (3)

```c
#include <stdio.h>

int main()
{
  int count;

  count = 0;
  printf("\nThe initial value of count is %d. ",count);
  count++;
  printf("\n count is now %d.", count);
  count++;
  printf("\n count is now %d.", count);
  count++;
  printf("\n count is now %d.", count);
  count++;
  printf("\n count is now %d.", count);

  return 0;
}
```

```
The initial value of count is 0.
 count is now 1.
 count is now 2.
 count is now 3.
 count is now 4.
```

Increment and Decrement Operators (4)

- When the ++ operator appears before a variable, it is called a prefix increment operator; when it appears after a variable, it is called postfix increment operator

```
k = ++n; // Prefix increment
  n = n + 1; // increment n first (step 1)
  k = n;     // assign n's value to k (step 2)

k = n++; // Postfix increment
  k = n;     // assign n's value to k (step 1)
  n = n + 1; // and then increment n (step 2)
```

Increment and Decrement Operators (5)

- Prefix decrement operator: the expression

```
k = --n;
```

  - first decrements the value of n by 1 before assigning the value of n to k

```
n = n - 1; // decrement n first (step 1)
k = n;     // assign n's value to k (step 2)
```

- Postfix decrement operator: the expression

```
k = n--;
```

  - first assigns the current value of n to k and then reduces the value of n by 1

```
k = n;     // assign n's value to k (step 1)
n = n - 1; // and then decrement n (step 2)
```

Topics

- Assignment
- Type Conversions
- Accumulating
- Operator Precedence
- Increment and Decrement Operators
- **printf() Function**
- scanf() Function
- Common Programming Errors

The `printf()` Function

- printf() formats data and sends it to the standard system display device (i.e., the monitor)
- Inputting data or messages to a function is called passing data (or is called passing arguments) to the function
  - printf("Hello there world!");
- Syntax (ไวยากรณ์ของภาษา): set of rules for formulating statements that are “grammatically correct” for the language
- กฎระเบียบต่าง ๆที่เกี่ยวข้องกับการเขียนชุดคำสั่ง โดยที่การเขียนชุดคำสั่งต่าง ๆต้องเขียนให้ถูกต้อง และสอดคล้อง กับหลักไวยากรณ์ของภาษานั้น ๆ

The `printf()` Function (2)

```
int printf(char *format, arg1, arg2, ...);
```

- Arguments
  - First argument of printf() must be a string
  - A string that includes a conversion control sequence, such as %d, is termed a control string
    - Conversion control sequences are also called conversion specifications and format specifiers

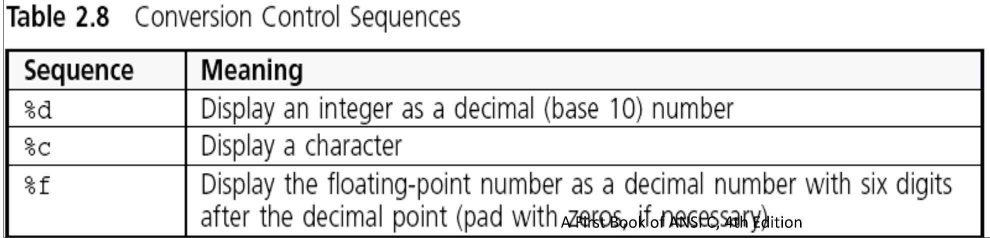


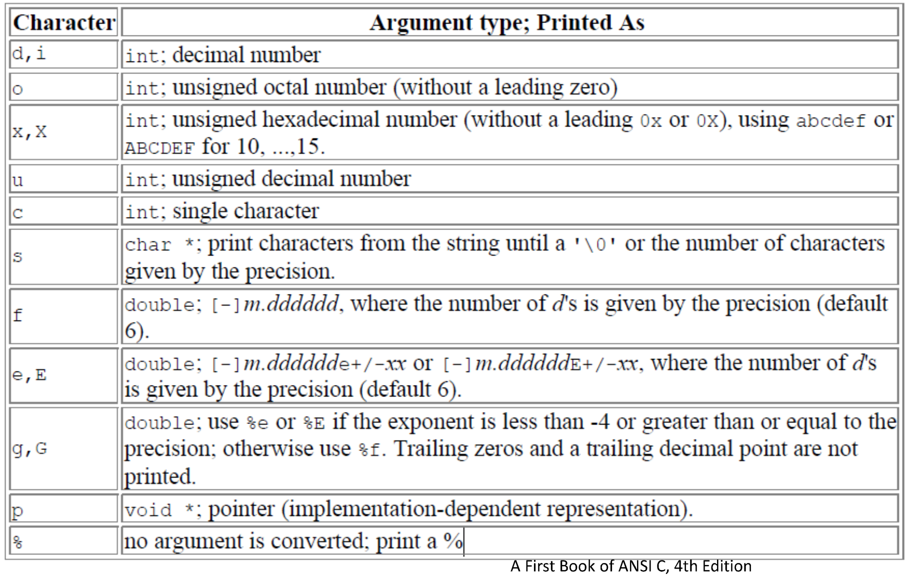
Table 2.8 Conversion Control Sequences

Sequence	Meaning
%d	Display an integer as a decimal (base 10) number
%c	Display a character
%f	Display the floating-point number as a decimal number with six digits after the decimal point (pad with zeros, if necessary)

The printf() Function (3)

Reference: The ANSI C Programming Language text book

***Table 7.1** Basic Printf Conversions*



Character	Argument type; Printed As
d,i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x,X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e,E	double; [-]*m.dddddd*e+/-*xx* or [-]*m.dddddd*E+/-*xx*, where the number of *d*'s is given by the precision (default 6).
g,G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

The printf() Function (4)

- **Control Sequence**
  - **printf() replaces a format specifier in its control string with the value of the next argument**

```
printf("The total of 6 and 15 is %d", 6 + 15);
```

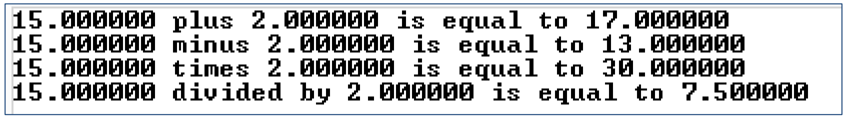
- **Output: The total of 6 and 15 is 21**

```
printf ("The sum of %f and %f is %f", 12.2,
                 15.754, 12.2 + 15.754);
```

- **Output: The sum of 12.200000 and 15.754000 is 27.954000**

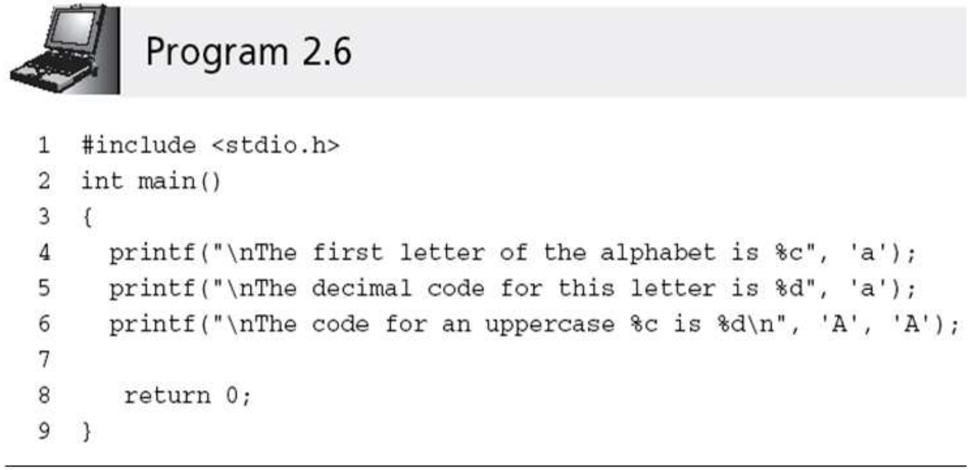
The printf() Function (5)

```
#include <stdio.h>

int main()
{
   printf("%f plus %f is equal to %f\n", 15.0, 2.0, 15.0+2.0);
   printf("%f minus %f is equal to %f\n", 15.0, 2.0, 15.0-2.0);
   printf("%f times %f is equal to %f\n", 15.0, 2.0, 15.0*2.0);
   printf("%f divided by %f is equal to %f\n", 15.0, 2.0, 15.0/2.0);

   return 0;
}
```



```
15.000000 plus 2.000000 is equal to 17.000000
15.000000 minus 2.000000 is equal to 13.000000
15.000000 times 2.000000 is equal to 30.000000
15.000000 divided by 2.000000 is equal to 7.500000
```

The printf() Function (6)



Program 2.6

```
#include <stdio.h>
int main()
{
  printf("\nThe first letter of the alphabet is %c", 'a');
  printf("\nThe decimal code for this letter is %d", 'a');
  printf("\nThe code for an uppercase %c is %d\n", 'A', 'A');

   return 0;
}
```

Format Modifiers

- **Left justification:** `printf("|%-10d|",59);`

  produces the display |59∧∧∧∧∧∧∧∧|
- **Explicit sign display:** `printf("|%+10d|",59);`

  produces the display |∧∧∧∧∧∧∧∧59|
- **Format modifiers may be combined**
  - **<u>%-+10d</u> would cause an integer number to both:**
    - **display its sign and**
    - **be left-justified in a field width of 10 spaces**
    - **The order of the format modifiers is not critical**
      - **%+-10d is the same**

Format Modifiers (2)

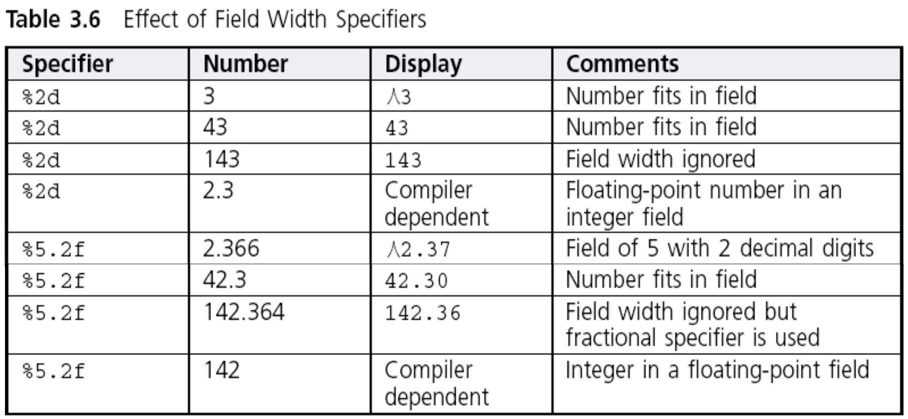
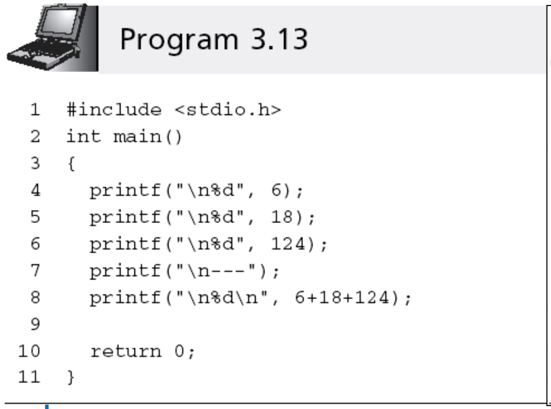


Table 3.6 Effect of Field Width Specifiers

Specifier	Number	Display	Comments
%2d	3	∧3	Number fits in field
%2d	43	43	Number fits in field
%2d	143	143	Field width ignored
%2d	2.3	Compiler dependent	Floating-point number in an integer field
%5.2f	2.366	∧2.37	Field of 5 with 2 decimal digits
%5.2f	42.3	42.30	Number fits in field
%5.2f	142.364	142.36	Field width ignored but fractional specifier is used
%5.2f	142	Compiler dependent	Integer in a floating-point field

Format Modifiers (3)

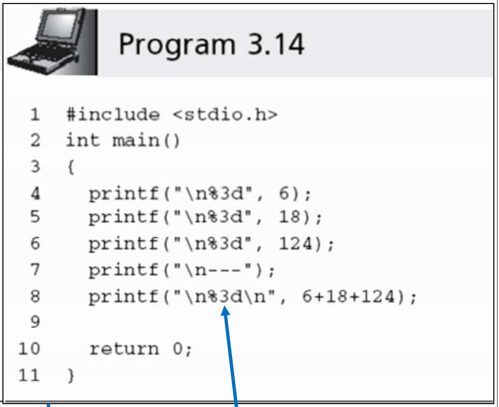


Program 3.13

```c
#include <stdio.h>
int main()
{
  printf("\n%d", 6);
  printf("\n%d", 18);
  printf("\n%d", 124);
  printf("\n---");
  printf("\n%d\n", 6+18+124);

  return 0;
}
```

Output

```
6
18
124
---
148
```



Program 3.14

```c
#include <stdio.h>
int main()
{
  printf("\n%3d", 6);
  printf("\n%3d", 18);
  printf("\n%3d", 124);
  printf("\n---");
  printf("\n%3d\n", 6+18+124);

  return 0;
}
```

Output

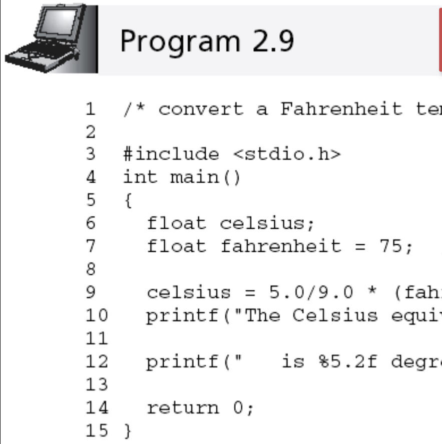
```
  6
 18
124
---
148
```

Field Width Specifier

Case Study: Temperature Conversion

- **Write and test a program that correctly converts the Fahrenheit temperature of 75 degrees into its Celsius equivalent.**

Celsius = 5/9(Fahrenheit – 32)



Program 2.9

```c
/* convert a Fahrenheit temperature to Celsius */

#include <stdio.h>
int main()
{
  float celsius;
  float fahrenheit = 75;   /* declaration and initialization */

  celsius = 5.0/9.0 * (fahrenheit - 32.0);
  printf("The Celsius equivalent of %5.2f degrees Fahrenheit\n",
                                                     fahrenheit);
  printf("   is %5.2f degrees\n", celsius);

  return 0;
}
```

Topics

- Assignment
- Type Conversions
- Accumulating
- Operator Precedence
- Increment and Decrement Operators
- printf() Function
- **scanf() Function**
- Common Programming Errors

The `scanf()` Function

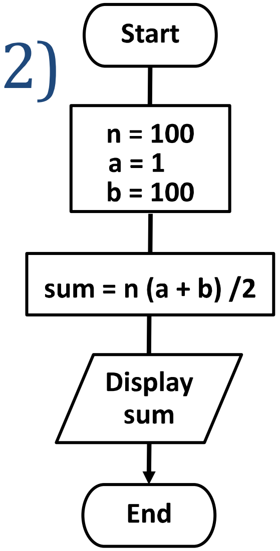
- **scanf() is used to enter data into a program while it is executing; the value is stored in a variable**
  - **It requires a control string as the first argument inside the function name parentheses, typically consists of conversion control sequences only**
- **scanf() requires that a list of variable addresses (&) follow the control string**
  - **scanf("%d", &num1); // d for integer**
  - **scanf("%f", &num2); // f for floating point**

หมายเหตุ: scanf() ยังสามารถใช้รับข้อมูลชนิดอื่น ๆได้อีกเช่น character, double เป็นต้น ซึ่งจะกล่าวถึงในบทเรียนถัดไป

The `scanf()` Function (2)

Practice 4: หาผลลัพธ์ของการบวกตัวเลข 1 ถึง 100

```
#include <stdio.h>
int main()
{
    int a, b, n, sum;

    printf("n = "); scanf("%d", &n);
    printf("a = "); scanf("%d", &a);
    printf("b = "); scanf("%d", &b);

    sum = n*(a+b)/2;
    printf("%d + %d + ... + %d = %d\n",
            a, a+1, b, sum);
    return 0;
}
```



Output-1

```
n = 100
a = 1
b = 100
1 + 2 + ... + 100 = 5050
```

Output-2

```
n = 98
a = 3
b = 100
3 + 4 + ... + 100 = 5047
```

The `scanf()` Function (3)

- **scanf() can be used to enter many values**

  `scanf("%f %f",&num1,&num2); //"%f%f" is the same`

- **A space can affect what the value being entered is when scanf() is expecting a character data type**

  `scanf("%c%c%c",&ch1,&ch2,&ch3);`

  - **Stores the next three characters typed in the variables ch1, ch2, and ch3;**
    - **if you type x y z, then:**
      - **x is stored in ch1,**
      - **a blank is stored in ch2, and**
      - **y is stored in ch3**

The scanf() Function (4)

- `scanf("%c %c %c",&ch1,&ch2,&ch3);` causes scanf() to look for three characters, each character separated by exactly one space
- When using scanf(), if a double-precision number is to be entered, you must use the %lf conversion control sequence
- scanf() does not test the data type of the values being entered
- In `scanf("%d %f", &num1, &num2)`,
  - if user enters 22.87
    - 22 is stored in num1 and
    - .87 is stored in num2

Caution: The Phantom Newline Character

Program 3.10

```
#include <stdio.h>
int main()
{
  char fkey, skey;

  printf("Type in a character: ");
  scanf("%c", &fkey);
  printf("The keystroke just accepted is %d", fkey);
  printf("\nType in another character: ");
  scanf("%c", &skey);
  printf("The keystroke just accepted is %d\n", skey);

  return 0;
}
```

Output

```
Type in a character: m
The keystroke just accepted is 109
Type in another character: The
keystroke just accepted is 10
```

10 is an Enter Key (New Line) (from keyboard buffer)

Caution: The Phantom Newline Character [2]

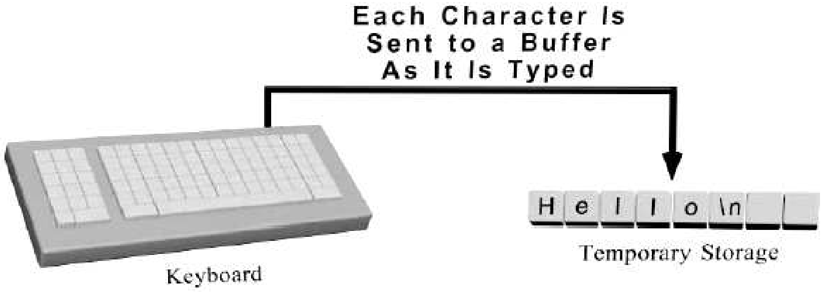


Figure 3.6 Typed keyboard characters are first stored in a buffer

Caution: The Phantom Newline Character [3]

Program 3.11

```
#include <stdio.h>
int main()
{
  char fkey, skey;

  printf("Type in a character: ");
  scanf("%c%c", &fkey, &skey); /* the enter code goes to skey */
  printf("The keystroke just accepted is %d", fkey);
  printf("\nType in another character: ");
  scanf("%c", &skey); /* accept another code */
  printf("The keystroke just accepted is %d\n", skey);

  return 0;
}
```

A First Look at User-Input Validation

Program 3.12

```
#include <stdio.h>
int main()
{
  int num1, num2, num3;
  double average;

  /* get the input data */
  printf("Enter three integer numbers: ");
  scanf("%d %d %d", &num1, &num2, &num3);

  /* calculate the average*/
  average = (num1 + num2 + num3) / 3.0;

  /* display the result */
  printf("\nThe avearge of %d, %d, and %d is %f\n",
                         num1, num2, num3, average);


  return 0;
}
```

A First Look at User-Input Validation [2]

- **As written, Program 3.12 is not robust**
- **The problem becomes evident when a user enters a non-integer value**

```
Enter three integer numbers: 10 20.68 20
The average of 10, 20, and -858993460 is -286331143.333333
```

- **Handling invalid data input is called user-input validation**
  - **Validating the entered data either during or immediately after the data have been entered**
  - **Providing the user with a way of reentering any invalid data**

To be continue in while loop

Practice

Celsius = 5/9(Fahrenheit – 32)

1. **Write and test a program that correctly converts the Fahrenheit temperature into its Celsius equivalent.**
   - **The Fahrenheit is input from keyboard**
2. **Write and test a program that correctly converts the Celsius temperature into its Fahrenheit equivalent.**
   - **The Celsius is input from keyboard**

Fahrenheit = *expression?*

Topics

- Assignment
- Type Conversions
- Accumulating
- Operator Precedence
- Increment and Decrement Operators
- printf() Function
- scanf() Function
- **Common Programming Errors**

Common Programming Errors

- Omitting the parentheses, (), after main

```
int main;
```

- Omitting or incorrectly typing the opening brace, {, that signifies the start of a function body

```
int main();
```

- Omitting or incorrectly typing the closing brace, }, that signifies the end of a function

```
int main()
{
```

- Misspelling the name of a function; for example, typing print() instead of printf()

```
print(“Hello World\n”);
```

Common Programming Errors [2]

- Forgetting to close a string passed to printf() with a double quote symbol

```
printf(“Hello World\n );
```

- Omitting the semicolon at the end of each executable statement

```
int x = 5, y = 2
```

- Forgetting to include \n to indicate a new line

```
printf(“Hello World”);
```

- Forgetting to declare all the variables used in a program

```
int x = 5, y = 367;
z = x + y;
```

- Storing an incorrect data type in a declared variable

```
int num;
num = 2.5;
```

Common Programming Errors [3]

- Using a variable in an expression before a value has been assigned to the variable

```
int x , y, z;
z = x * y / 2;
```

- Dividing integer values incorrectly

```
int x = 3, y;
y = x / 2; // ได้ผลลัพธ์เท่ากับ 1 เศษถูกปัดทิ้ง
```

- Mixing data types in the same expression without clearly understanding the effect produced

```
double x = 5.2, z;
int y;
z = x % y * 3;
```

Common Programming Errors [4]

- Not including the correct conversion control sequence in printf() function calls for the data types of the remaining arguments

```
printf("Result of  %f + %d = %d", 6.2, 5, 6.2/3);
```

- Not closing the control string in printf() with a double quote symbol followed by a comma when additional arguments are passed to printf()

```
printf("Result of  %f + %d = %d, 6.2, 5, 6.2/3);
```

- Forgetting to separate all arguments passed to printf() with commas

```
printf("Result of  %.2f + %d = %.2f", 6.2  5
6.2/3);
```