บทที่ 1 บทนำ

1.1 ประวัติความเป็นมา

C ถูกพัฒนาขึ้นโดย Dennis Ritchie ของ Bell Telephone Laboratories, Inc. (AT&T Bell Laboratories ในปัจจุบัน) และมีรากฐานมาจากภาษา BCPL (Basic Combined Programming Languuage ; by Martin Richards of Cambridge University) และ B (Ken Thompson, AT&T Bell Laboratories) ซึ่งพัฒนาโดย Bell Laboratories เช่นกัน C เป็นที่รู้จักของบุคคลภายนอก Bell Lab เป็นครั้งแรกเมื่อปี 1978 โดยการตีพิมพ์เอกสารเผยแพร่ภาษา C ของ Brian Kernighan และ Dennis Ritchie หรือที่รู้จักกันดีในนาม "K&R C"

หลังจากนั้น C ก็ได้รับความนิยมในหมู่นักเขียนโปรแกรมทั่วไป เนื่องจากมีจุดเด่นพิเศษหลายประการ จนทำให้มีการผลิตโปรแกรม C Compiler ออกมาสู่ท้องตลาด และมีการนำภาษา C ไปใช้พัฒนา Software ต่างๆ ซึ่งเดิมเขียนด้วยภาษาอื่น เพื่อเป็นการเพิ่มประสิทธิภาพ (Efficiency) และเพิ่มความสามารถในการใช้ (Portability) เพราะง่ายต่อการนำไปใช้กับหลายๆ Platform หรือหลายๆ Operating System

ในระยะแรก ภาษา C จะไม่มีมาตรฐานที่แน่นอน ภาษา C ที่หลายๆ บริษัทผู้ผลิต C Compiler กำหนดขึ้นจะต่างจาก K&R C ทั้งนี้ ก็เพื่อง่ายต่อการพัฒนา C Compiler และเพิ่มความสะดวกในการใช้ให้กับผู้ใช้ มาตรฐานของภาษา C ที่ใช้กันในปัจจุบัน จะเป็นมาตรฐานที่ถูกกำหนดโดย ANSI (American National Standards Institute) หรือเรียกสั้นๆ ว่า ANSI C ซึ่งผู้ผลิต C Compiler จะยึดเอา ANSI C เป็นมาตรฐานในการผลิต โดยจะเพิ่ม Function ต่างๆ ที่อำนวยความสะดวกในการเขียน เช่น Function เกี่ยวกับ Graphics Function สำหรับเขียนโปรแกรมบน Windows เป็นต้น ดังนั้น ถ้าเราเขียนโปรแกรมภาษา C โดยใช้มาตรฐานของ ANSI C ไม่ว่าจะใช้ C Compiler ตัวใดในการ Compile เราก็สามารถทำการ Compile โปรแกรมดังกล่าวได้

ในช่วงปีทศวรรษที่ 90 ได้มีความพยายามในการปรับปรุงและเพิ่มขีดความสามารถของภาษา C ทำให้เกิดเป็นมาตรฐานภาษา C ฉบับใหม่ ซึ่งมีชื่อเรียกว่า C99 โดยมีคุณลักษณะเพิ่มเติมอันได้แก่

- เพิ่มชนิดข้อมูลใหม่ เช่น complex bool เป็นต้น
- เพิ่มจำนวนไฟล์ header มาตรฐาน เช่น complex.h stdbool.h เพื่อสนับสนุนการใช้งานชนิดข้อมูลที่เพิ่มเข้ามา
- ใช้เครื่องหมายแสดง comment แบบเดียวกับ C++ กล่าวคือ เครื่องหมาย //
- สามารถแทรกส่วนการประกาศตัวแปรให้อยู่ระหว่างประโยคคำสั่งได้ เช่น อนุญาตให้ทำการประกาศตัวแปรภายในวงเล็บหลัง for ได้

จุดเด่นของภาษา C

- ถูกพัฒนาขึ้นเพื่อใช้เป็น System Programming Language (UNIX operating System) แต่ปัจจุบัน ใช้ในงานทั่วไป
- เป็นภาษาที่สนับสนุนการเขียนโปรแกรมในลักษณะโครงสร้าง (Structured Programming)
- Low-level Language กล่าวคือ สามารถประมวลผลข้อมูลในระดับที่ใกล้เคียงกับภาษาเครื่อง (Machine Language) ได้ ซึ่งได้แก่ การอ้างอิงโดยใช้ Pointer การคำนวณแบบ Bitwise เป็นต้น (จัดว่าอยู่ระหว่างกลางระหว่าง High-level Language กับ Machine Language)
- ประสิทธิภาพในการทำงานสูง ทำงานได้รวดเร็ว เมื่อเทียบกับภาษา High-level อื่นๆ

- High Portability กล่าวคือ สามารถนำโปรแกรมที่เขียนด้วยภาษา C ไปใช้กับเครื่องอื่นๆ ที่มี C Compiler ได้ โดยไม่ต้องทำการแก้ไขโปรแกรม หรือแก้ไขเพียงเล็กน้อย
- Procedural Language (หรือ Imperative Language) กล่าวคือ มีลักษณะเป็นการสั่งงานเป็นขั้นตอน โดยมุ่งเน้นที่การเขียนชุดคำสั่งสำหรับให้คอมพิวเตอร์ทำการประมวลผล ซึ่งแตกต่างจากวิธีการโปรแกรมเชิงวัตถุ (object-oriented programming)

1.2 โครงสร้างของโปรแกรมภาษา C

โปรแกรมภาษา C ประกอบด้วยส่วนต่างๆ ดังต่อไปนี้

ส่วน Preprocessor

ส่วน *Preprocessor* จะอยู่ส่วนแรกสุดของโปรแกรม มีหน้าที่จัดการเกี่ยวกับตัวโปรแกรมก่อนที่จะถูก Compile เช่น การรวมไฟล์ (File Inclusion) การกำหนดค่าคงที่ต่างๆ (Constant Definition) เป็นต้น ซึ่งในการใช้ Function มาตรฐานที่มีอยู่ใน Library จะต้องทำการรวมไฟล์ที่กำหนดเกี่ยวกับ Function ที่จะใช้ และค่าของตัวแปรต่างๆ ที่เกี่ยวข้อง โดยทั่วไป ไฟล์ที่ต้องทำการรวม ได้แก่ ไฟล์ชื่อ stdio.h ซึ่งจะกำหนดเกี่ยวกับ Functon มาตรฐานที่ใช้ในการรับค่า และแสดงผล อันได้แก่ scanf, printf, gets เป็นต้น ยกตัวอย่าง เช่น

```
#include <stdio.h>
#define PI 3.141592654
```

ส่วน Function

ส่วน *Function* เป็นส่วนที่สำคัญที่สุดของโปรแกรม เพราะจะเป็นส่วนที่กำหนดการทำงานทั้งหมดของโปรแกรม ซึ่ง *Function* ในโปรแกรมภาษา C จะมีอย่างน้อยที่สุด 1 Function คือ Function *main* เป็น Function ที่ควบคุมการทำงาน และการเรียกใช้ Function อื่นๆ ในกรณีที่มีหลาย Function (พูดง่ายๆ คือ เป็นเสมือนผู้จัดการโปรแกรม โดย Function อื่นๆ นั้นเป็นพนักงานทั่วไป) Function แบ่งออกเป็นส่วนต่างๆ ดังนี้

1. ***Function header*** ประกอบด้วย

- *function-name* คือชื่อของ Function จะต้องตั้งชื่อไม่ให้ซ้ำกับ Function ที่มีอยู่เดิมหรือ Function ที่มีอยู่ใน Standard Library เพราะมิฉะนั้น อาจจะไม่สามารถเรียกใช้ Function เหล่านั้นได้
- *arguments* เป็นค่าที่สำหรับป้อนให้กับ Function เพื่อใช้ในการคำนวณ หรือการทำงานต่างๆ เช่น Function สำหรับหาค่า sin ก็ต้องป้อนค่า *x* เป็นต้น *argument* จะมีหรือไม่มีก็ได้ ขึ้นอยู่กับ Function และมีได้มากกว่า 1 สำหรับ Function *main* โดยทั่วไปจะไม่มี argument แต่ในกรณีที่ต้องการให้ตัวโปรแกรมรับค่าตอนที่สั่งงานที่ Command-line (ลักษณะเดียวกับในกรณีที่ใช้คำสั่งใน COMMAND PROMPT เช่น คำสั่ง COPY ก็ต้องระบุ [*source*] [*destination*] ซึ่ง [*source*] [*destination*] นี้ ในภาษาคอมพิวเตอร์ จะเรียกว่าเป็น “Command-line argument”) Function *main* จะต้องมี argument
- *return-value-data-type* โดยทั่วไป Function จะต้องมีการคืนค่าใดค่าหนึ่งกลับมา ซึ่งค่าที่คืนมานั้น จะต้องกำหนดว่า เป็นค่าชนิดไหน (character, integer, floating point หรืออื่นๆ) การกำหนดชนิดของค่าที่คืนมานั้น กำหนดได้โดยใส่ *return-value-data-type* ไว้ที่หน้า *function-name* ในบาง Function เช่น Function สำหรับแสดงค่า ซึ่งเราไม่ต้องการให้มีการคืนค่าใดๆ กลับมา ให้ระบุ *return-value-data-type* เป็น *void* โดยเฉพาะ Function *main* ซึ่งปกติ จะไม่มีการคืนค่า

ส่วนประกอบที่ว่านี้ จะถูกเขียนเรียงกันในลักษณะ

```
[return-value-data-type] function-name( arguments )
```

ยกตัวอย่าง เช่น

```
int sum(int x, int y)
```

2. ***declaration*** เป็นส่วนที่ใช้สำหรับกำหนดตัวแปร และชนิดของตัวแปรที่ใช้ในการคำนวณ ยกตัวอย่าง เช่น

```
int   a,b,c,d ;
float x, y, z ;
char  insert_char ;
```

3. ***statement*** (ประโยคคำสั่ง) เป็นส่วนที่ใช้สำหรับสั่งงาน เช่น สั่งให้คำนวณค่าต่างๆ สั่งให้รับค่าต่างๆ สั่งให้แสดงผล เป็นต้น ยกตัวอย่าง เช่น

```
a = 2 ;
z = x + y ;
printf("Hello") ;
for ( i = 0; i < 10; i++ ) { printf( "%d ", i ) ; }
```

ส่วน Comment

ส่วน *Comment* เป็นส่วนที่ใช้สำหรับแสดง Message ต่างๆ เพื่อให้ง่ายต่อการทำความเข้าใจของผู้เขียน และผู้อ่านโปรแกรม โดยจะเขียนอยู่ระหว่างสัญลักษณ์ /* กับ */ Compiler จะไม่ทำการแปลข้อความที่อยู่ระหว่างสัญลักษณ์ทั้งสองนี้ นอกจากนี้ มาตรฐาน C99 ยังอนุโลมให้สามารถเขียน Comment ทีละบรรทัดโดยใช้สัญลักษณ์ // (slash 2 ตัว) นำหน้า Comment

หมายเหตุ

1. โปรแกรม C Compiler ในปัจจุบัน มักจะกำหนดให้ Function ทุกอัน จะต้องมีการคืนค่า (มีคำสั่ง *return*) ดังนั้น ถ้าหากไม่ได้ระบุ *return-value-data-type* มักจะเกิด Warning ยกตัวอย่างเช่น ถ้าหากไม่กำหนด *return-value-data-type* ของ Function *main* เป็น *void* แล้ว ไม่มีประโยคคำสั่ง return ตอน Compile จะเกิด Warning Message ว่า Function should return a value in function main() ซึ่งไม่ก่อให้เกิดปัญหาในการรันโปรแกรม โปรแกรมก็ยังสามารถทำงานอย่างปกติได้ แต่ถ้าหากเราไม่ต้องการให้เกิด Warning ขึ้น เราสามารถทำได้โดยการกำหนด *return-value-data-type* เป็น *void* ซึ่งเป็นการบอก Compiler ให้รู้ว่า จะไม่มีการคืนค่า
2. ส่วน Declaration และ Statement จะอยู่ระหว่างสัญลักษณ์ { และ } และแต่ละ declaration หรือ statement จะถูกขั้นด้วยสัญลักษณ์ ; (semi-colon)

ตัวอย่างที่ 1.1 Very Popular Basic C Program : Hello World Program

```
/*      Hello World Program :
        for displaying "Hello World" on the monitor    */
#include <stdio.h>                /* Preprocessor */
int main()                        // function heading
{
        printf("Hello World\n");   /* statement : instruct the program to
                                   print the message "Hello World" */
        return 0;
}
```

ผลการรันโปรแกรม

```
Hello World
```

ตัวอย่างที่ 1.2 Basic Arithmetic Operation

```
/*     Basic Arithmetic Operation :  For calculating sum of 2 integers    and
sum of 2 floating point numbers  */
#include <stdio.h>
int main()                                        /* function heading *?
{
       /* declarations */
       int    a,b,c ;                    /* integer variables */
       float  x,y,z ;                    /* floating point variables */
       /* statements */
       a = 2; b = 3;                     /* assignment statements */
       c = a + b ;                       /* sum of 2 integers */

       x = 2.0 ;  y = 3.0 ;        /* assignment statements */
       z = x + y ;                       /* sum of 2 floating point */
       /* Display result */
       printf("c = %d z = %f\n", c, z) ;
       return 0;
}
```

ผลการรันโปรแกรม

```
c = 5 z = 5.000000
```

ตัวอย่างที่ 1.3 Area of a circle

```
/*  Program to calculate area of a circle */
#include <stdio.h>
int main()                                        /* function heading */
{
       float  radius, area ;             /* variable declaration */

       printf(" Radius = ? ") ;          /* output statement (PROMPT) */
       scanf("%f", &radius) ;            /* input statement */
       area = 3.14159 * radius * radius ; /* assignemt statement */
       printf("Area = %f\n", area) ;     /* output statement */
       return 0;
}
```

ผลการรันโปรแกรม

```
Radius = ? 5
Area = 78.539749
```

ข้อควรระวังเกี่ยวกับการเขียนโปรแกรมภาษา C

- ในภาษา C จะไม่มีการกำหนดบรรทัดอย่างแน่นอน ยกเว้นแต่ในส่วนของ *Preprocessor*

  ยกตัวอย่าง เช่น ประโยคคำสั่ง 2 ประโยคข้างล่างนี้ มีค่าเท่ากัน

  ```
  printf("%c%c%c reversed is %c%c%c",char1,char2,char3,char3,char2,char1);
  printf("%c%c%c reversed is %c%c%c\n",
            char1, char2, char3,
            char3, char2, char1) ;
  ```

  หรือสามารถเขียนหลายประโยคไว้ในบรรทัดเดียวกันได้ เช่น

  ```
      x = 2.0 ;  y = 3.0 ;
  ```

  สังเกตว่า แต่ละประโยคจะสิ้นสุดด้วย ;

- ไม่ควรใช้ keyword (reserved word , ดู Appendix A) , operator, หรือ standard function name เป็นชื่อตัวแปร (variable name) โดยทั่วไป ควรใช้เฉพาะตัวพยัญชนะภาษาอังกฤษ (a-z, A-Z) ตัวเลข (0-9) และสัญลักษณ์ _ (underscore) เท่านั้นในการตั้งชื่อตัวแปร ชื่อฟังก์ชัน และชื่อชนิดข้อมูล (มีชื่อเรียกรวมว่า "identifier")

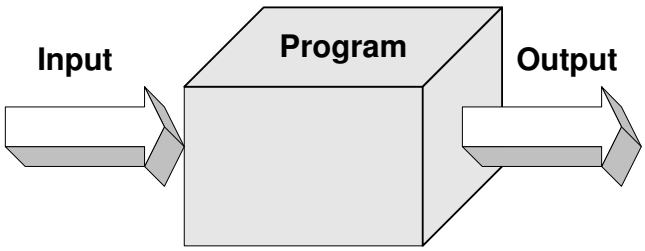
- ตัวแปรที่ใช้ในโปรแกรมทั้งหมดจะต้องได้รับการ Declaration ก่อนจะใช้ (ยกเว้นตัวแปรที่ได้รับการ Declaration เรียบร้อยแล้วใน standard header file, e.g. stdio.h หรือใน header file ที่ทำการรวมเข้ามา)
- ตัวพิมพ์ใหญ่ ตัวพิมพ์เล็ก C Compiler จะมองตัวแปร Name, name และ NAME เป็นคนละตัวแปร พูดง่ายๆ คือ C Compiler จะทำการแยกแยะระหว่างตัวพิมพ์ใหญ่กับตัวพิมพ์เล็ก (ไม่เหมือนกับ Windows หรือ DOS ที่ถือว่า เป็นตัวเดียวกัน) ตัวพิมพ์ใหญ่จะใช้แต่เฉพาะเป็นชื่อของตัวแปรที่มีค่าคงที่ที่ได้รับการกำหนดนิยามไว้ในส่วนในส่วน *Preprocessor* (Constant Macro)
- การย่อหน้า และการเว้นช่องว่าง ดูตัวอย่างต่อไปนี้

Bad Example	Good Example

1.1. `for(i=0;i<20;i++){result+=i;}`
1.2. `for ( i=0 ; i<20 ; i++ )`
`{`
`result += i ;`
`}`
	2.1
	`i = 0 ;`
	`while( i < 20 )`
	`{`
	`result+=i;`
	`i++;`
	`}`
	2.2
	`i = 0 ;`
	`while ( i < 20 )`
	`{`
	`result += i ;`
	`i++ ;`
	`}`

1.3 ฟังก์ชันอินพุต/เอาต์พุตมาตรฐาน (Standard I/O functions)

โดยทั่วไป การใช้โปรแกรมจะเป็นไปตามรูปที่ 1.1 กล่าวคือ ผู้รันโปรแกรมจะทำการป้อนข้อมูลสำหรับใช้ในการคำนวณให้กับโปรแกรม หลังจากนั้น ก็เป็นหน้าที่ของโปรแกรมที่จะทำการคำนวณ แล้วแสดงผลออกมาให้ผู้รันโปรแกรมทราบ ถ้าหากว่า ไม่สามารถป้อนข้อมูลให้กับโปรแกรมได้ ก็จะเป็นโปรแกรมที่มีขีดจำกัดในการใช้งานค่อนข้างต่ำ (หรืออีกนัยหนึ่งคือ มี Generality ต่ำ) และถ้ายิ่งไม่สามารถแสดงผลได้ ก็คงจัดเป็น *JUNK PROGRAM* (*โปรแกรมขยะ*) ที่ใช้ประโยชน์อะไรไม่ได้



<u>รูปที่ 1.1</u> การใช้โปรแกรม

ดังนั้น ในการเขียนโปรแกรม จะต้องรู้จักวิธีการสั่งให้โปรแกรมทำการรับข้อมูล และแสดงผลการคำนวณออกมา ซึ่งในทุกภาษา ก็ได้มีการเตรียมสิ่งเหล่านี้ไว้ให้ เนื่องจากถ้าไม่มี ก็จะเป็นภาระของผู้เขียนโปรแกรม จะต้องทำการพัฒนาเอง อันเป็นงานที่ไม่ง่ายนัก เพราะต้องทำการเขียนโปรแกรมให้ติดต่อกับระบบปฏิบัติการเพื่อจะได้ใช้ Interrupt ได้ ในภาษา C สิ่งเหล่านี้ จะมีให้ในรูปของฟังก์ชัน (ดังที่ได้กล่าวแล้วว่ารูปแบบของโปรแกรมย่อยในภาษา C มีเพียงแบบเดียวคือ ฟังก์ชัน) ในที่นี้ จะกล่าวถึง Standard Input / Standard Output และฟังก์ชันอินพุต/เอาท์พุตมาตรฐานของภาษา C ที่ใช้กับ Standard Input และ Standard Output

Standard Input

Standard Input หรือย่อสั้นๆ ว่า *stdin* มีหน้าที่รับข้อมูลจากผู้ใช้ โดยทั่วไป ถ้าไม่ได้กำหนดเงื่อนไขอะไรเป็นพิเศษ (ในภาษาคอมพิวเตอร์ ใช้คำว่า Default) จะหมายถึง การป้อนข้อมูลทางคีย์บอร์ด เป็นวิธีการป้อนข้อมูลที่พื้นฐานและใช้กันมากที่สุด วิธีนี้ จะทำให้เกิดการโต้ตอบระหว่างโปรแกรมกับผู้ใช้โปรแกรมได้

Standard Output

Standard Output หรือย่อสั้นๆ ว่า *stdout* มีหน้าที่แสดงผล โดยทั่วไป Default ของ Standard Output จะเป็นจอมอนิเตอร์ กล่าวคือ ถ้าไม่ได้กำหนดอะไรเป็นพิเศษ โปรแกรมจะแสดงผลออกทางหน้าจอมอนิเตอร์

Standard Input Function

เนื่องจากในการป้อนค่าทางคีย์บอร์ด ผู้ใช้สามารถป้อนค่าได้เฉพาะเป็นตัวอักษร ดังนั้น การที่คอมพิวเตอร์จะสามารถรับข้อมูลชนิดต่างๆ ไม่ว่าจะเป็นตัวอักษร จำนวนเต็ม หรือเลขทศนิยม จำเป็นที่จะต้องทำการแปลงตัวอักษรที่ป้อนโดยผู้ใช้ให้เป็นชนิดของข้อมูลต่างๆ ที่ต้องการ *Standard Input Function* เป็นชื่อเรียกรวมของฟังก์ชันที่ทำหน้าที่ดังกล่าวนี้ ได้แก่ *getchar printf gets* เป็นต้น

Standard Output Function

เนื่องจาก Text Mode ของจอมอนิเตอร์แสดงได้แต่เพียงตัวอักษร ดังนั้น การที่จะแสดงข้อมูลทางจอมอนิเตอร์ได้จะต้องทำการแปลงข้อมูลที่อยู่ในหน่วยความจำ ซึ่งเป็นเลขฐานสอง ให้กลายเป็นตัวอักษรเสียก่อน *Standard Output Funciton* เป็นชื่อเรียกรวมของฟังก์ชันที่ทำหน้าที่ดังกล่าวนี้ ได้แก่ *putchar scanf puts* เป็นต้น

getchar

getchar มีไว้สำหรับรับข้อมูลจาก Standard Input ทีละ 1 ตัวอักษร

รูปแบบ : `int getchar(void)`

Header File : stdio.h

ค่าที่คืน : คืนค่าตัวอักษรตัวต่อไปที่รับจาก Standard Input ในกรณีที่อ่านถึงท้ายสุดของไฟล์หรือเกิดข้อผิดพลาดขึ้น จะคืนค่า EOF (ย่อมาจาก **E**nd **O**f **F**ile โดยมากมักจะมีค่าเท่ากับ -1 เพื่อให้มีหลักประกันว่า จะไม่ซ้ำกับค่าของตัวอักษรตัวอื่นๆ)

วิธีการเรียกใช้ : variable = getchar() ;

ตัวอย่าง :
```
int  a ;
a = getchar() ;
```
หมายถึงรับข้อมูลจาก stdin ทีละ 1 ตัวอักษร แล้วไปเก็บเป็นค่าของตัวแปร *a*

putchar

putchar มีไว้สำหรับแสดงผลออกทาง Standard Output ทีละ 1 ตัวอักษร

รูปแบบ : int putchar(int character) ;

Header File : stdio.h

ค่าที่คืน : คืนค่าตัวอักษรที่แสดง หรือค่า EOF กรณีที่เกิดข้อผิดพลาดขึ้น

วิธีการเรียกใช้ : โดยปกติ จะไม่มีการเอาตัวแปรมารับค่าที่คืนมา ในลักษณะดังนี้

putchar(variable) ;

ตัวอย่าง : putchar(a) ;

หมายถึงแสดงตัวอักษรที่มีค่าเท่ากับค่าของตัวแปร *a* ทาง stdout

ตัวอย่างที่ 1.4 Usage of getchar and putchar

```
1:    /* Program for getting one character and displaying it */
2:    #include <stdio.h>
3:    int main()
4:    {
5:           int    a;
6:           printf("Enter any character \n") ;
7:           a = getchar() ;     /* Get a character */
8:           putchar(a) ;    /* Print the input character */
9:           putchar('\n') ;  /* Print newline character */
10:          return 0;
11:   }
```

ผลการรันโปรแกรม

```
Enter any character
A
A
```

scanf

scanf สำหรับใช้รับข้อมูลจาก Standard Input โดยสามารถกำหนดรูปแบบการรับข้อมูลได้

รูปแบบ : int scanf(const char* format_string, address1, ..., addressN) ;

Header File : stdio.h

ค่าที่คืน : คืนจำนวนค่าที่อ่านได้ หรือค่า EOF กรณีที่เกิดข้อผิดพลาดขึ้น

วิธีการเรียกใช้ : `scanf( "%d,%d,%d", &a, &b, &c ) ;`

หมายถึงรับจำนวนเต็ม 3 ค่า ที่ขั้นด้วยเครื่องหมาย , (Comma) แล้วเก็บเป็นค่าของตัวแปร a b และ c ตามลำดับ

ในการเรียกใช้ฟังก์ชันนี้ ต้องการ Argument หลักๆ อยู่ 2 ตัวคือ Format String สำหรับกำหนดรูปแบบการรับข้อมูลและ Address ของหน่วยความจำที่จะใช้เป็นที่เก็บค่า (พูดตามภาษา C ก็คือ Pointer) ตามจำนวนค่าที่จะรับเข้ามา เนื่องจากในการป้อนค่าจาก stdin จะป้อนได้โดยตัวอักษรเท่านั้น ดังนั้น Format String จะเป็นสิ่งสำคัญที่จะบอกให้โปรแกรมรู้ว่า จะอ่านค่าจาก stdin แล้วตีความหมายอย่างไร จึงจะถูกต้อง

Format String จะประกอบด้วย

- ช่องว่าง (whitespace) ซึ่งอาจจะเป็นได้ทั้งช่องว่างธรรมดา TAB หรือการขึ้นบรรทัดใหม่ นอกจากกรณีที่มีช่องว่าง อยู่ในวงเล็บ [ ] แล้ว ช่องว่างจะเป็นตัวกำหนดให้ฟังก์ชันอ่านข้ามช่องว่างที่ต่อเนื่องกันทั้งหมด จนกว่าจะถึงตัวอักษรที่ไม่ใช่ช่องว่าง
- ตัวอักษรที่ไม่ใช่ whitespace หรือ % ที่จะต้องใช้ในการเปรียบเทียบกับตัวอักษรที่รับเข้ามา ยกตัวอย่าง เช่น สมมุติว่า เราต้องการให้ผู้ใช้ป้อนข้อมูลในลักษณะ 4, 5.5, 8.7 เราจะต้องใส่ตัวอักษร , (comma) ไว้ใน Format String
- Conversion Code เป็น Code สำหรับควบคุมการรับค่าของฟังก์ชัน ประกอบด้วย
    1. เครื่องหมาย %

2. เครื่องหมาย * ที่ใช้ในการสั่งให้ฟังก์ชันทำการ skip ข้อมูลอันนั้น กล่าวคือ จะไม่ทำการอ่านข้อมูลอันนั้น
3. ตัวเลขจำนวนเต็ม เป็นตัวเลขสำหรับกำหนดความกว้างสูงสุดของข้อมูลที่จะทำการอ่าน
4. Code ดูในตารางที่ 1.1 ประกอบ

ข้อ 1 และ 4 เป็นสิ่งจำเป็น ในขณะที่ข้อ 2 และ 3 ไม่มีก็ได้

โดยทั่วไป ฟังก์ชันจะทำการ skip ช่องว่าง ยกเว้นกรณีที่ Code เป็น %[ ] และ %c ในกรณีที่ใช้ %*n*c ฟังก์ชันจะทำการอ่านตัวอักษรตัวถัดไป จำนวน *n* ตัว รวมทั้งช่องว่างด้วย และจะไม่มีตามเติมตัวอักษร '\0' (NUL) ต่อท้าย ในขณะที่ถ้าใช้ %*n*s ฟังก์ชันจะเริ่มต้นจากการหาตัวอักษรที่ไม่ใช่ช่องว่าง แล้วทำการอ่านไป จนกว่าจะรับตัวอักษรจนครบ *n* ตัว หรือเจอช่องว่าง

printf

printf สำหรับใช้แสดงข้อมูลออกทาง Standard Output โดยสามารถกำหนดรูปแบบการแสดงข้อมูลได้

รูปแบบ : int scanf(const char* format_string, arg1, ..., argN) ;

Header File : stdio.h

ค่าที่คืน : คืนจำนวนตัวอักษรที่แสดง หรือจำนวนเต็มลบ กรณีที่เกิดข้อผิดพลาดขึ้น

วิธีการเรียกใช้ : printf("%d,%d,%d", a, b, c) ;

หมายถึงแสดงค่าของตัวแปรชนิดจำนวนเต็ม 3 ตัว คือ a, b และ c ขั้นด้วยเครื่องหมาย , (comma)

printf("Hello World\n");

แสดงข้อความว่า Hello World แล้วขึ้นบรรทัดใหม่

ในการเรียกใช้ฟังก์ชันนี้ ต้องการ Argument หลักๆ อยู่ 2 ตัวคือ Format String สำหรับกำหนดรูปแบบการแสดงข้อมูลและตัวแปรที่ต้องการแสดงค่า ในกรณีที่ไม่มีตัวแปรที่ต้องการแสดงค่า ฟังก์ชันจะแสดงข้อความนั้นออกทาง Standard Output เนื่องจาก TEXT MODE ของจอมอนิเตอร์สามารถแสดงได้เฉพาะตัวอักษรเท่านั้น ดังนั้น Format String จะเป็นตัวบอกให้ฟังก์ชันรู้ว่า จะต้องทำการแปลงค่าของตัวแปรที่จัดเก็บในหน่วยความจำในรูปของเลขฐานสอง เป็นตัวอักษรอย่างไร เพื่อที่จะสามารถแสดงออกทาง stdout ได้อย่างถูกต้อง

ตารางที่ 1.1 การใช้ code ใน Format String ของฟังก์ชัน scanf

Code	*Interpretation*	*Example of Input*	*Corresponding Argument Must Be*
c	ตัวอักษร	p	address of char
s	ชุดตัวอักษร (String)	pie	address of char
d	เลขจำนวนเต็มฐานสิบ แปลงเป็น int	27563	address of int
hd	เลขจำนวนเต็มฐานสิบ แปลงเป็น short int	-6693	address of short
ld	เลขจำนวนเต็มฐานสิบ แปลงเป็น long int	2965372	address of long
o	เลขจำนวนเต็มฐานแปด แปลงเป็น int	55273	address of int
ho	เลขจำนวนเต็มฐานแปด แปลงเป็น short int	5232	address of short
lo	เลขจำนวนเต็มฐานแปด แปลงเป็น long int	-6273123	address of long
x, X	เลขจำนวนเต็มฐานสิบหก แปลงเป็น int	FFFAB	address of int
hx, hX	เลขจำนวนเต็มฐานสิบหก แปลงเป็น short int	-b2e	address of short
lx, lX	เลขจำนวนเต็มฐานสิบหก แปลงเป็น long int	2d3d4d	address of long
i	เลขจำนวนเต็มฐานสิบ, ฐานแปด (นำหน้าด้วย 0) ฐานสิบหก (นำหน้าด้วย 0x) แปลงเป็น int	065273	address of int
hi	เลขจำนวนเต็ม แปลงเป็น short int	0xB2E	address of short
li	เลขจำนวนเต็มฐาน แปลงเป็น long int	2957827	address of long
u	unsigned int	52723	address of int
hu	unsigned short	42327	address of short
lu	unsigned long	369527832	address of long
e, f, g, E, G	เลขทศนิยมแปลงเป็น float	4.2328e+03	address of float
le, lf, lg, lE, lG	เลขทศนิยมแปลงเป็น double	3.141592654	address of double
Le, Lf, Lg, Le, LG	เลขทศนิยมแปลงเป็น long double	3.14159265358979324	address of long double
p	address	(implementation dependent)	address of a pointer to void
[*chars*]	string ที่มีเฉพาะตัวอักษรที่อยู่ใน *chars*	edit (ในกรณีที่ *chars* เป็น [abcdefghit])	address of char
[^*chars*]	string ที่ไม่มีตัวอักษรที่อยู่ใน *chars*	edit (ในกรณีที่เป็น [^xyz])	address of char
%	ตัวอักษร % ในอินพุต	%	(none)

Format String ที่ใช้กับฟังก์ชัน printf ประกอบไปด้วย

- ตัวอักษรทั่วไปที่ต้องการให้ฟังก์ชันแสดง
- Conversion Code ซึ่งประกอบด้วยส่วนต่อไปนี้ ตามลำดับ
    1. เครื่องหมาย %
    2. Optional Flag (ตารางที่ 1.2)
    3. ตัวเลขจำนวนเต็มบวก แสดงความกว้างของข้อมูลที่จะแสดง (ตารางที่ 1.4)
    4. เครื่องหมาย . (period) สำหรับแยก Precision ออกจากความกว้างของข้อมูล

5. ตัวเลขจำนวนเต็มบวก สำหรับกำหนด precision (ความละเอียด)
6. Code (ตารางที่ 1.3)

ในกรณีที่ใช้ %f, %e, %E, %g หรือ %G ถ้าไม่ได้กำหนด Precision ฟังก์ชันจะแสดงตัวเลขหลังจุดทศนิยมจำนวน 6 หลักโดยอัตโนมัติ

ตารางที่ 1.2 การใช้ Flag ใน Format String ของฟังก์ชัน printf

Flag	***การแปลความหมาย***
-	Left-justify
+	เริ่มต้นด้วยเครื่องหมาย + หรือ -
blank	ถ้า signed integer ไม่เริ่มต้นด้วย + หรือ - ให้เติมช่องว่างลงไป 1 ช่อง
#	สำหรับ Code o เริ่มต้นด้วย 0
	สำหรับ Code x เริ่มต้นด้วย 0x
	สำหรับ Code X เริ่มต้นด้วย 0X
	สำหรับ Code e, E, f, g และ G ใช้จุดทศนิยมแบบเลขฐานสิบ
	สำหรับ Code g และ G ไม่มีการตัด 0 ที่ต่อท้าย
0 (zero)	เติมเลข 0 เข้าไปด้านซ้าย

ตารางที่ 1.3 การใช้ code ใน Format String ของฟังก์ชัน printf

Code	*Interpretation*	*Example of Output*	*Corresponding Argument Must Be*
c	ตัวอักษร	p	char, short, int
s	ชุดตัวอักษร (String)	pie	address of char
d, i	เลขจำนวนเต็มฐานสิบ	-999	char, short, int
ld, li	เลขจำนวนเต็มฐานสิบ เขียนแบบ long	299999999	long
o	เลขจำนวนเต็มฐานแปด	7037	char, short, int
lo	เลขจำนวนเต็มฐานแปด เขียนแบบ long	7255577	long
x, X	เลขจำนวนเต็มฐานสิบหก	fe5, FE5	char, short, int
lx, lX	เลขจำนวนเต็มฐานสิบหก เขียนแบบ long ใช้ตัวอักษร a-f	2d3d5f, 2D3D5F	long
u	เลขจำนวนเต็มฐานสิบแบบ unsigned	280732	unsigned char, unsigned short, unsigned int
lu	เลขจำนวนเต็มฐานสิบแบบ unsigned long	3772957283	unsigned long
e, E	เลขทศนิยม float, double แสดงในรูป m.nnnnnne±xx, m.nnnnnnE±xx	3.1415920e+03 3.1415920E+03	float, double
f	เลขทศนิยม float, double แสดงในรูป m.nnnnnn	2.718282	float, double
g, G	เลขทศนิยม float, double แสดงในรูปที่สั้นที่สุดใน d, e (หรือ G) หรือ f	33.900000	float, double
Le, LE, Lf, Lg, LG	เหมือนกับ e, E, f, g, G ของเลขทศนิยม float, double เพียงแต่ใช้กับ long double	2.9723E+04	long double
p	address	(implementation dependent)	pointer to void
%	แสดงตัวอักษร %	%	(none)

ตารางที่ 1.4 ตัวอย่างการใช้ Format String ของฟังก์ชัน printf และผลที่ได้ ในกรณีที่กำหนดให้ตัวแปร val, val2, fval และ *s มีค่าเท่ากับ -15, 87, 43.718, และ "computer" ตามลำดับ

FORMAT	EXPRESSION	OUTPUT
"%15d"	val	-15
"%-15d"	val	-15
"%15.5d"	val	-00015
"%15d"	val2	87
"%+15d"	val2	+87
"%-15d"	val2	87
"%15.5d"	val2	00087
"%f"	fval	43.718994
"%15.0f"	fval	44
"%-15.0f"	fval	44
"%+15.0f"	fval	+44
"%10.5f"	fval	43.71800
"%.5f"	fval	43.71800
"%15x"	val	fffffff1
"%#15x"	val	0xfffffff1
"%15s"	s	computer
"%-15s"	s	computer

ตัวอย่างที่ 1.5 ตัวอย่างการใช้ standard I/O function (printf & scanf)

```
1:    /* Example of using scanf and printf */
2:    #include <stdio.h>
3:    int main()
4:    {
5:       int    a, b, c ;
6:       printf("ENTER 2 number(use format:number1,number2) \n") ;
7:       scanf("%d,%d", &a, &b);
8:       /* Read 2 integers from keyboard
9:       using format number1, number2 */
10:      c = a + b ;           /* Calculate sum of 2 integers */
11:      printf("%d + %d = %d\n", a, b, c) ; /* Print the result */
12:      return 0;
13:   }
```

ผลการรันโปรแกรม

```
ENTER 2 number (use format : number1, number2)
23, 37
23 + 37 = 60
```

แบบฝึกหัด

1. เขียนโปรแกรมสำหรับแสดงข้อความว่า "Learning C language is fun."
2. เขียนโปรแกรมสำหรับคำนวณหาผลบวก ลบ คูณ หารของเลข 2 จำนวน
3. เขียนโปรแกรมสำหรับคำนวณหาพื้นที่ของรูป DONUT
4. เขียนโปรแกรมสำหรับคำนวณหาพื้นที่ของสามเหลี่ยม
5. เขียนโปรแกรมสำหรับคำนวณหาพื้นที่ของสี่เหลี่ยมผืนผ้า และสี่เหลี่ยมคางหมู
6. เขียนโปรแกรมสำหรับแปลงหน่วยไมล์เป็นกิโลเมตร
7. เขียนโปรแกรมสำหรับแปลงหน่วยปอนด์เป็นกิโลกรัม
8. เขียนโปรแกรมที่รับค่ามุมเป็นหน่วยองศา แล้วแปลงเป็นหน่วยเรเดียน