บทที่ 2
ตัวแปรและตัวดำเนินการ

โดยปกติตัวแปรและตัวคงที่ในภาษา C ก็คล้าย ๆ กันกับที่มีใช้ในภาษาอื่น ๆ แต่ภาษา C มีจำนวนตัวดำเนินการ (operator) มากกว่า เช่น bitwise shift, prefix, postfix และอื่น ๆ

2.1 ตัวแปร

ตัวแปรในภาษา C มีได้หลายแบบ โดยมากจะไม่ต่างจากตัวแปรในภาษาอื่น อาจแตกต่างกันบ้างในรายละเอียด เช่น รายละเอียดของตัวแปรที่เป็นจำนวนเต็ม - (integer) สามารถจำแนกรายละเอียดเป็น short, long, unsigned และ long

float และโดยปกติเราจะต้องกำหนดลักษณะ (declaration) ของตัวแปรก่อนเสมอ โดยรูปแบบของการกำหนดลักษณะปรากฏดังนี้

type variable-name;

หรือ

type variable-name, variable-name . . . .

คำว่า type หมายถึงชนิดของตัวแปร คำว่า variable-name หมายถึงชื่อตัวแปร หากมีหลายตัวให้แยกจากกันด้วยเครื่องหมายจุลภาค (comma ,) และปิดท้ายด้วยเครื่องหมาย-อัฒภาค (Semicolon ;) โดยปกติชื่อตัวแปรจะต้องเริ่มต้นด้วยตัวอักษร หรือเครื่องหมายขีดเส้นใต้(_) ต่อจากนั้นจึงเป็นตัวอักษร หรือตัวเลขหรือขีดก็ได้ โดยเราจะตั้งชื่อไว้กี่อักขระก็ได้แต่คอมไพเลอร์ (compiler) จะรับได้เพียง 8 อักขระแรก และถ้าเผอิญเราหลงลืมมิได้ระบุประเภทของตัวแปรเครื่องจะกำหนดให้เป็น int เสมอ ตัวอย่างการกำ-หนดลักษณะตัวแปรกระทำได้ดังนี้

int ;	กำหนดให้ตัวแปรชื่อ i เป็นตัวแปรแบบ integer
int i,j ;	กำหนดให้ตัวแปรชื่อ i และ j เป็นตัวแปรแบบ integer
char c ;	กำหนดให้ตัวแปรชื่อ c เป็นตัวแปรแบบ character
float f ;	กำหนดให้ตัวแปรชื่อ f เป็นตัวแปรแบบ float (real)
double d ;	กำหนดให้ตัวแปรชื่อ d เป็นตัวแปรแบบ long float
short int i ;	กำหนดให้ตัวแปรชื่อ i เป็นตัวแปรแบบ short integer (เรามุ่งหมายเพื่อประหยัดพื้นที่)
long int j ;	กำหนดให้ตัวแปรชื่อ j เป็นตัวแปรแบบ long integer(เรามุ่งหมายเพื่อเพิ่มพื้นที่ให้ตัวแปรรับเลขจำนวนสูง ๆ ได้)
unsigned int b ;	กำหนดให้ตัวแปรชื่อ b เป็นตัวแปรแบบ unsigned integer

มุ่งหมายเพื่อให้ตัวแปร b มีค่าเป็นบวกเสมอ หรือกำหนดให้ significant bit งดเว้นทำหน้าที่กำหนดเครื่องหมาย (ดูเรื่อง bit, byte)

`unsigned char c ;` กำหนดให้ตัวแปรชื่อ c เป็นตัวแปรแบบ unsigned character (มุ่งหมายเพื่อให้ตัวแปร c ที่เมื่อ convert เป็น integer variable แล้วมีค่าเป็นบวกเสมอ)

ตัวอย่างการกำหนดลักษณะตัวแปรในภาษาอื่นปรากฏดังนี้

BASIC

I%	integer:
C$	character:
F	float:
D#	double:

(these are derived from the suffix on the name of the variable)

FORTRAN

INTEGER I	integer
REAL F	float
DOUBLE D	double

(no character in FORTRAN-66)

PASCAL

I : INTEGER;	integer
C : CHAR;	character
F : REAL;	float

(no double in PASCAL)

PL/I

DECLARE I FIXED BINARY (15,0);	integer
DECLARE C CHAR(1);	character
DECLARE F FLOAT BINARY(31);	float
DECLARE D FLOAT BINARY(63);	double

COBOL

02 I PICTURE 99999.	integer
02 C PICTURE X.	character
02 F PICTURE 9999V999.	float
02 D PICTURE 999999999999V999.	double[3]

อนึ่งตัวแปรต่าง ๆ ดังที่กล่าวมามีขนาดและพิสัย (ช่วงข้อมูลที่เป็นตัวเลข) ดังนี้

ชนิดตัวแปร	ขนาด	พิสัย
char	1 ไบท์	- 128 ถึง 127 หรือ 0 ถึง 255 แล้วแต่คอมไพเลอร์
int	2 ไบท์	- 32768 ถึง 32767
float	4 ไบท์	- 1.7×10^{38} ถึง 1.7×10^{38} (ความถูกต้อง 6 ตำแหน่ง)
double	8 ไบท์	- 1.7×10^{38} ถึง 1.7×10^{38} (ความถูกต้อง 16 ตำแหน่ง)
short int	2 ไบท์	-32768 ถึง 32767
long int	4 ไบท์	- 2147483648 ถึง 2147483647
unsigned int	2 ไบท์	- 0 ถึง 65535

2.2 ตัวคงที่

ตัวคงที่ในภาษา C มีลักษณะและแบบเช่นเดียวกับตัวคงที่ในภาษาอื่น ๆ ดังนี้

ก. integer constant หมายถึงตัวคงที่ที่เป็นเลขจำนวนเต็มเช่น 2, 20, 32

ข. long constant หมายถึงตัวคงที่ที่เป็นเลขจำนวนเต็มที่มีขนาด 4 ไบท์เท่ากับ long int โดยเราจะต้องผนวกอักษร L หรือ I ไว้หลังตัวเลขจำนวนเต็มเพื่อแสดงให้คอมไพเลอร์ทราบว่าเป็น long constant เช่น 2L, 20L, 32L, 2I,

20I, 32I อย่างไรก็ตามหากไม่ผนวกอักษร L หรือ I ไว้ถ้าข้อมูลที่บันทึกมีพิสัยเกินกว่า -32768 ถึง 32767 คอมไพเลอร์ก็จะจัดการให้ตัวคงที่นั้นเป็น long constant เอง เช่น 567893

ค. float constant หมายถึงตัวคงที่ที่เก็บข้อมูลในพิสัยเดียวกับ double Variable (8 ไบท์) คือระหว่าง -1.7×10^{38} ถึง 1.7×10^{38} (ความถูกต้อง 16 ตำแหน่ง) เช่น 32.3, 12.5E-7

ง. octal constant หมายถึงตัวคงที่แบบ integer constant ที่มีค่าเป็นเลขฐาน 8 โดยเราจะต้องเติมเลข 0 (ศูนย์) หน้าจำนวนเต็มใด ๆ เพื่อแสดงว่าเป็นจำนวนเต็มในฐาน 8 เช่น 034

จ. hexadecimal constant หมายถึงจำนวนเต็มในฐาน 16 โดยจะต้องเติม 0x หรือ 0x (ศูนย์เอกซ์) ข้างหน้าจำนวนเต็มในฐาน 16 เช่น OXFD, OxFE

นอกเหนือไปจากตัวคงที่ที่กล่าวมาซึ่งจัดว่าเป็นจำนวนคงที่ (numeric constant) ยังมีตัวคงที่อีกชนิดหนึ่งเรียกว่า Character constant ตัวคงที่ชนิดนี้อาจจะเป็นอักขระเดี่ยว ๆ เช่น A, B, C, ..., Z, a, b, c, ...z อักขระหลายตัวเช่น NUL, SOH, STX, VS, SP รวมตลอดถึงเครื่องหมายและตัวเลข เช่น !, %, @,= ,0, 1, ..., 9 และอื่น ๆ ซึ่งเป็นอักขระที่เมื่อแปลงกลับเป็นค่าในรหัสแอสกี้จะมีค่าเป็นตัวเลข ตัวคงที่ที่เป็น Character constant นั้นเวลาใช้เราต้องเขียนเครื่องหมายคำพูดเดี่ยว (single guote) คลุมไว้เช่น 'A', 'Z', '=' ซึ่งมีค่าใน ASCII เป็น 65, 90 และ 61 ตามลำดับ สำหรับ ASCII codeซึ่งแสดงการเปรียบเทียบอักขระต่าง ๆ ไปในระบบเลขฐาน 2, 8, 10 และ 16 รวมทั้งประโยชน์ในลักษณะอื่น(ดูหมายเหตุด้านขวาตาราง) ปรากฏดังนี้

Decimal	Hex	CHR	Decimal	Hex	CHR	Decimal	Hex	CHR
000	00	NUL	036	24	$	072	48	H
001	01	SOH	037	25	%	073	49	I
002	02	STX	038	26	&	074	4A	J
003	03	ETX	039	27	'	075	4B	K
004	04	EOT	040	28	(	076	4C	L
005	05	ENQ	041	29	)	077	4D	M
006	06	ACK	042	2A	*	078	4E	N
007	07	BEL	043	2B	+	079	4F	O
008	oa	BS	044	2C	,	080	50	P
009	09	H-r	045	2D	-	081	51	Q
010	0A	LF	046	2E	.	082	52	R
011	R?	VT	047	2F	/	083	53	S
012	0C	FF	048	30	0	084	54	T
013	0D	CR	049	3,	1	085	55	U
014	0E	so	050	32	2	086	56	V
015	0F	SI	051	33	3	087	57	W
016	10	DLE	052	34	4	088	58	X
017	11	DC1	053	35	5	089	59	Y
018	12	DC2	054	36	6	090	5A	Z
019	13	Cc3	055	37	7	091	5B	[
020	14	Lx4	056	38	8	092	5C	\
021	15	NAK	057	39	9	093	5D	]
022	16	SYN	058	3A	:	094	5E	^
023	17	ETB	059	3B	;	095	5F	_
024	18	CAN	060	3C	<	096	60	`
025	19	EM	061	3D	=	097	61	a
026	1A	SUB	062	3E	>	098	62	b
027	1B	ESCAPE	063	3F	?	099	63	c
028	1C	FS	064	40	@	100	64	d
029	1D	GS	065	41	A	101	65	e
030	1E	RS	066	42	B	102	66	f
031	1F	US	067	43	C	103	67	g
032	20	SPACE	068	44	D	104	68	h
033	21	!	069	45	E	105	69	i
034	22	"	070	46	F	106	64	j
035	23	#	071	47	G	107	6B	k
108	6C	l	115	73	s	122	7A	z
109	6D	m	116	74	t	123	7B	{
110	6E	n	117	75	u	124	7c	|
111	6F	o	118	76	v	125	7D	}
112	70	p	119	77	w	126	7E	~
113	71	q	120	78	x	127	7F	DEL
114	72	r	121	79	y			

Decimal	Hex	Character
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	Pt
159	9F	ƒ

Decimal	Hex	Character
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▒
178	B2	▒
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Decimal	Hex	Character
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▉
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Decimal	Hex	Character
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
236	EB	δ
236	EC	∞
237	ED	ø
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
246	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	•
250	FA	·
251	FB	√
252	FC	n
253	FD	2
254	FE	■
255	FF	(BLANK)

อนึ่งในภาษา C จะพบว่าเราสามารถใช้ escape sequence (ประกอบด้วย backslash คือ \ เรียกว่า escape character ผสมกับอักษรตัวเล็ก) เพื่อให้เครื่องทำหน้าที่บางอย่างได้ ดังนี้

escape sequence	ความหมาย
'\n'	เลื่อนบรรทัด (LF), newline character
'\t'	ตั้งระยะ (HT) , tab character
'\\'	backslash
'\''	เครื่องหมายคำพูดเดี่ยว
'\0'	ศูนย์
'\b'	ถอยหลัง (BS)
'\f'	เลื่อนกระดาษทั้งแผ่น (FF)
'\r'	cariage return
'\"'	เครื่องหมายคำพูดคู่

หากไม่มี escape sequence ใช้และประสงค์จะให้พิมพ์ข้อมูลอื่น ๆ ที่สนใจให้ใช้ backslash รวมกับเลขฐาน 8 เช่น '\001' หมายถึงเลข 1 '\007' หมายถึง bell, '\011' หมายถึง horizontal tab ก็ได้ (ดูรหัสแอสกี้)

นอกจากนี้ยังมีตัวคงที่อีกแบบหนึ่งเรียกว่า string constant ตัวคงที่แบบนี้จะถูกเก็บไว้ในส่วนความจำในรูปของ character constant ซึ่งจะมีสัญญาณให้ทราบว่าเป็นสตริงด้วยการปิดท้ายด้วย \0' เช่น (ขอให้สังเกตว่าเราใช้เครื่องหมายคำพูดคู่ กรณี string constant และใช้เครื่องหมายคำพูดเดี่ยว กรณีcharacter constant)

" ABCDE " คือ string constant ที่ประกอบด้วยอักขระ 'A', 'B', 'C', 'D', 'E', และ \ 0'

" " คือ string constant ที่ประกอบด้วยเฉพาะ '\ 0'

ตัวอย่างเพื่อแสดงการเปรียบเทียบตัวคงที่ในภาษาอื่นปรากฏดังนี้

BASIC

32	integer constant
32.3, 12.5E-7	float constants
"A"	single character constant
"ABCDEF"	string constant

FORTRAN

32	integer constant
32.3, 12.5E-7	float constants
1HA	character constant in data statement
6HABCDEF	character constants in data statement

PASCAL

32	integer constant
32.3, 12.5E-7	float constants
'A'	single character constant
'ABCDEF'	string constant

PL/I

32	integer constant
32.3, 12.5E-7	float constants
'011100'B	octal constant represented by binary constant
'11111110'B	hexadecimal constant represented by binary constant
'A'	single character constant
'ABCDE'	string constant

COBOL

32	integer constant
32.3, 12.5E-7	float constants
"A"	single character constant
"ABCDE"	string constant

2.3 การกำหนดค่าเริ่มต้น (initialization)

การกำหนดค่าเริ่มต้นนั้นควรจะกระทำพร้อมกันไปกับการกำหนดลักษณะตัวแปร แต่จะไม่ทำเช่นนี้ก็ได้ การไม่กำหนดค่าเริ่มต้นให้แก่ตัวแปรจะมีผลให้ตัวแปรนั้นมีค่าเท่ากับ 0 หรือมีค่าเท่ากับค่าที่ค้างอยู่ในแอดเดรสนั้น จากการทำงานรอบก่อนหน้านั้น (garbage) ทั้งนี้ขึ้นอยู่กับ storage class หากเราจะกำหนดค่าเริ่มต้นพร้อมทั้งกำหนดลักษณะตัวแปรพร้อมกันไปเราจะกระทำได้ดังนี้

`int i = 5`	หมายถึงกำหนดค่าเริ่มต้นให้แก่ตัวแปร i ที่เป็น integerให้เท่ากับ 5
`float f = 32.5`	หมายถึงกำหนดค่าเริ่มต้นให้แก่ตัวแปร f ที่เป็น - ตัวแปรแบบ float ให้เท่ากับ 32.5
`char c = 'A' ;`	หมายถึงกำหนดค่าเริ่มต้นให้แก่ตัวแปร c ที่เป็น character variable ให้เท่ากับค่าของ A ใน ASCII-code (A=65)

storage class ของตัวแปรประกอบด้วย

1) auto

การเตรียมพื้นที่ส่วนความจำ (memory location) ให้แก่ตัวแปร - automatic (เรียกย่อว่า auto) นั้นจะกระทำเฉพาะเมื่อมีโปรแกรมถูกอ่านผ่านเข้ามา เมื่อทำงานตามโปรแกรมเสร็จ พื้นที่ในส่วนนั้นก็จะพร้อมให้โปรแกรมอื่นผ่านเข้ามาใช้ได้ เครื่องจะถือว่า (เป็น default) ตัวแปรทุกตัวในโปรแกรมเป็นตัวแปร auto ยกเว้นจะกำหนดให้เป็นแบบอื่นไว้เป็นการเฉพาะ เราควรกำหนดค่าเริ่มต้นให้แก่ตัวแปร auto เพื่อให้ตัวแปรนี้มีค่าเท่ากับค่าเริ่มต้นนี้ทุกครั้งที่ทำงาน หากไม่กำหนดค่าเริ่มต้นให้ตัวแปร auto จะมีค่าเท่ากับค่าที่ค้างอยู่เดิม (garbage) automatic storage

จึงเป็นพื้นที่ในส่วนความจำที่รับค่าตัวแปรเข้ามาเก็บรักษาไว้จนกว่าจะมีค่าใหม่ก็จะถูกเก็บไว้ในที่นั้นแทนโดยอัตโนมัติ

2) static

การเตรียมพื้นที่ในส่วนความจำให้แก่ตัวแปร static จะถูกจองหรือสำรองไว้เมื่อเริ่มคอมไพล์และจะจองที่ไว้เช่นนั้น การกำหนดค่าเริ่มต้นของตัวแปร static จะมีผลให้ตัวแปรมีค่าเป็นอย่างอื่นได้ หากไม่กำหนดค่าเริ่มต้นให้เครื่องจะถือว่ามีค่าเท่ากับ 0 (default)

3) register

ตัวแปร register ทำหน้าที่คล้ายตัวแปร auto การระบุตัวแปรว่าเป็น register เป็นการแจ้งให้คอมไพเลอร์จัด machine register แทน memory location ให้แก่ตัวแปร การกระทำดังกล่าวจะทำให้ได้ผลลัพธ์รวดเร็วมาก อย่างไรก็ตามหากเรากำหนดให้ตัวแปรจำนวนมากเป็นตัวแปรแบบ register จนทำให้ machine register ไม่เพียงพอกับความต้องการกรณีเช่นนี้ตัวแปรที่เหลือจะถูกจัดให้เป็นตัวแปรแบบ auto

4) **external**

ตัวแปรแบบ external คือตัวแปรที่เรากำหนดไว้นอกฟังก์ชัน โดยปกติตัวแปรภายนอกนี้ เราจะต้องกำหนดค่าเริ่มต้นให้ชัดเจนหากไม่กำหนดตัวแปรภายนอกจะมีค่าเริ่มต้นเท่ากับ 0 (default) เสมอ อนึ่งเนื่องจากการคอมไพล์ source file นั้นเราสามารถแยกทำได้ เมื่อประสงค์จะเชื่อมโยงเข้าด้วยกันก็กระทำได้ด้วยการอาศัย

อาร์กิวเมนต์หรือตัวแปรภายนอก การกำหนดให้ตัวแปรเป็นตัวแปรภายนอกเราจะกำหนดไว้ในไฟล์เพียงไฟล์เดียว ถ้าไฟล์อื่นต้องการใช้ตัวแปรภายนอกนั้นบ้างให้เรียกใช้เป็น externเช่น extern int x เพื่อแจ้งให้คอมไพเลอร์ทราบว่า นี่คือตัวแปรเดียวกันกับ x ที่เป็นตัวแปรภายนอก ให้นำมูลค่าที่เก็บไว้ในที่ชื่อ x มาใช้

2 . 4 ตัวดำเนินการ (operator)

ตัวดำเนินการในภาษา C โดยส่วนใหญ่แล้วจะคล้ายๆกันกับตัวดำเนินการในภาษาอื่น แต่ในภาษา C อาจมีตัวดำเนินบางลักษณะที่ผิดแผกไปจากภาษาเหล่านั้น เช่น prefix,postfix,assignment,conditional operator

1) ตัวดำเนินการคณิต (arithematic operator)

ตัวดำเนินการคณิตประกอบด้วย +,-,*,/ เช่นเดียวกับภาษาอื่น หากตัวแปรที่อยู่คนละด้านของเครื่องหมายเหล่านี้ตัวแปรตัวใดตัวหนึ่งจะต้องได้รับการแปลง (convert)ไปเป็นตัวแปรแบบเดียวกันเสียก่อน เช่น float variable แปลงเป็น double variable เป็นต้น(ดูเรื่อง conversion ในตอน 2.5)

การบวกลบคูณหารในภาษา C กระทำเช่นเดียวกันกับในภาษาอื่น เช่น 5+i หมายถึงนำ 5 ไปบวกกับค่าของตัวแปร i 22.3 *f หมายถึงนำ 22.3 คูณกับค่าของตัวแปร f k/3 หมายถึงนำค่าของ k มาหารด้วย 3 และ x-y หมายถึงนำค่าของ y มาหักออกจากค่าของ x สำหรับการนำจำนวนเต็มไปหารจำนวนเต็ม ผลหารจะเป็นจำนวนเต็มและเศษเศษในรูป modulus operator(%) เช่น 22/3 จะมีค่าเป็นจำนวนเต็มเท่ากับ 7 เศษเท่ากับ 22%3 (ซึ่งมีค่าเท่ากับ 1)

ตัวดำเนินการคณิต ในภาษา c มี unary เช่นกันกับในภาษาอื่น แต่ไม่มีการยกกำลัง (exponentiation operator) เหมือนในภาษาอื่น

2) ตัวดำเนินการเปรียบเทียบ (relational operator)

ในภาษา c มีตัวดำเนินการเปรียบเทียบเหมือนกับในภาษาอื่นคือ

= =	เท่ากับ, เท่ากัน
! =	ไม่เท่ากับ(! หมายถึง not != ก็คือ< > ในภาษาเบสิคนั่นเอง)
>	มากกว่า
<	น้อยกว่า
> =	มากกว่าหรือเท่ากับ
< =	น้อยกว่าหรือเท่ากับ

ซึ่งผลการเปรียบเทียบจะออกมาเป็น 1 ถ้าจริงและเป็น 0 ถ้าเท็จหรือไม่จริง เช่น 5<3 มีค่าเป็น 0 3<5 มีค่าเป็น 1 5 ==5 มีค่าเป็น 1 3 ==5 มีค่าเป็น 0 i <=3 มีค่าเป็น 1 ถ้า i มีค่าน้อยกว่า 3 และเป็น 0 ถ้า i มีค่าตั้งแต่ 4 เป็นต้นไป

3) ตัวดำเนินการตรรก (logical operator)

การดำเนินการ (operate) ในภาษา c เกี่ยวกับตรรกจะกระทำจากซ้ายไปขวาโดยมีปฏิบัติการ 2 แบบคือ & & (หมายถึง AND) และ || (หมายถึง inclusive OR) ซึ่งจะมีค่าเป็นเป็น 0 ถ้าเท็จและเป็น 1 ถ้าจริง กล่าวคือ

(1) & & (AND)

กรณี AND นี้เครื่องจะตรวจทางซ้ายของ & & ก่อนว่าเป็น 0 หรือไม่ หากพบว่ามีค่าเป็น 0 เครื่องจะไม่อ่านทางขวา และถือว่าผลของการดำเนินการ(operate)

เป็นเท็จคือให้ค่าเท่ากับ 0 ทันที ดังนี้

ด้านซ้าย	ด้านขวา	มูลค่าของผลลัพธ์
0	ไม่ต้องประเมิน	0
nonzero	0	0
nonzero	nonzero	1

เช่น 5 && 3 มีค่าเท่ากับ 1 5 && 0 มีค่าเท่ากับ 0

2) || (OR)

กรณี OR นี้เครื่องจะตรวจทางซ้ายก่อนว่าเป็น 0 หรือไม่ ถ้าพบว่าค่าทางซ้ายมือของ || มีค่าไม่เป็น 0 (nonzero) เครื่องจะไม่อ่านค่าทางขวา และถือว่าผลของการปฏิบัติการเป็นจริงคือมีค่าเท่ากับ 1 ทันที ดังนี้

ด้านซ้าย	ด้านขวา	มูลค่าของผลลัพธ์
0	0	0
0	nonzero	1
nonzero	ไม่ต้องประเมิน	1

เช่น 5||3 มีค่าเท่ากับ 1 5||0 มีค่าเท่ากับ 1 i || j มีค่าเท่ากับ 0 ถ้าทั้ง i และ j มีค่าเป็น 0 มิเช่นนั้นแล้วจะมีค่าเป็น 1

<u>หมายเหตุ</u> คำว่าไม่เป็น 0 หรือ "nonzero" หมายถึง true 0 หมายถึง false

(3)! (negation)

negation คือปฏิบัติการแย้งซึ่งจะกลับความหมายของมูลค่าต่าง ๆ เป็น-ตรงข้าม คือ ค่าที่ไม่เป็น 0 จะถูกกลับค่าให้เป็นเท็จ (คือ 0) และกลับค่าที่เป็น 0 ให้เป็นจริง (คือ 1) เช่น

! 5 มีค่าเป็น 0 ! 0 มีค่าเป็น 1 ! i มีค่าเป็น 0 ถ้า i เป็นจำนวนที่ไม่เป็น 0 และเป็น 1 ถ้า i เป็น 0

4) **bitwise** operator

bitwise operator ใช้สำหรับปิด-เปิด หรือทดสอบบิทใด ๆ ในกรณีที่ตัวแปรเป็น integer variable bitwise operator คือการปฏิบัติการทางตรรก (AND, OR, XOR) รวมถึงการย้าย bit string ไปทางซ้าย-ขวา และส่วนเติมเต็ม (หรือ NOT) เช่น

bit a	NOT bit a
1	0
0	1

bit a	bit b	bit a & bit b (& หมายถึง AND)
0	0	0
0	1	0
1	0	0
1	1	1

bit a	bit b	bit a | bit b (| หมายถึง OR)
0	0	0
0	1	1
1	0	1
1	1	1

bit a	bit b	bit a ∧ bit b (∧ หมายถึง XOR)
0	0	0
0	1	1
1	0	1
1	1	0

หมายเหตุ ∧ หรือ XOR คือ exclusive OR ซึ่งถือว่าสิ่งต่าง ๆ จะเป็นจริงต่อเมื่อสิ่งหนึ่งเป็นจริงคือ T ∧ F ได้ T และ F ∧ T ได้ T

ดังนั้น สตริงของบิทจึงเกิดขึ้นจากการนำเอาบิทต่อบิทของแต่ละสตริงมา - Operate กับตามกฎของ bitwise operator เช่น 11010 & 10111 กระทำดังนี้

bit a	bit b	bit a & bit b
1	1	1
1	0	0
0	1	0
1	1	1
0	1	0

ดังนั้น 11010 & 10111 ก็คือ 10010

หมายเหตุ เครื่องหมาย & หมายถึง AND | หมายถึง OR ∧ หมายถึง exclusive OR << หมายถึงเคลื่อนบิทไปทางซ้าย >> หมายถึงเคลื่อนบิทไปทางขวา ขอให้สังเกตว่า bitwise operator ใช้เครื่องหมายโดยเฉพาะ AND และ OR ต่างกันกับ logical operator เล็กน้อย อนึ่งการเคลื่อนบิทไปทางซ้าย เราจะเติม 0 ไปด้านท้ายของbit stringเท่ากับจำนวน bit ที่เคลื่อนเช่น 00001111 << 2 หมายถึงเคลื่อน bit string 00001111 ไปทางซ้าย 2 bit ผลลัพธ์คือ 00111100 ถ้าเคลื่อน bit stringไปทางขวาให้เติม 0 เข้าด้านหน้า bit string เท่ากับจำนวน bit ที่เคลื่อนไป (ถ้าเป็น logical shift)เช่น 01001001 >> 1 หมายถึงเคลื่อน bit string 01001001 ไปทางขวา 1 bit ผลลัพธ์คือ 00100100 (กรณี logical shift) และใช้ sign bit เติมเต็มด้านหน้า (ถ้าเป็น arithmetic shift)เช่น 01001001 >>1 ผลลัพธ์คือ 00100100 และ 10011000 >>1 ผลลัพธ์คือ 11001100

ตัวอย่าง bitwise operator ปรากฏดังนี้

		Equivalent in Bits (Binary)	
Expression	*Value*	*Expression*	*Value*
1 | 2	3	00000001 | 00000010	00000011'
0xFF&0x0F	0x0F	11111111 & 00001111	00001111'
0x33 | oxcc	0xFF	00110011 | 11001100	11111111'
0x0F<<2	0x3C	00001111<<2	00111100'
0x1C>>1	0x0E	00011100>>1	000111O'
~x03	0xFC	~0000000000000011	1111111111111100'

5) ตัวดำเนินการกำหนดค่า (assignment operator)

ในภาษา C เราใช้เครื่องหมายเท่ากับคือ = เป็นตัวดำเนินการกำหนดค่า ทำหน้าที่สำเนา (Copy) ค่าทางขวามือของเครื่องหมายเท่ากับ (=) ลงไปบนแอดเดรส (address) หรือตัวแปรที่อยู่ด้านซ้ายมือ ผลการดำเนินการจะปรากฏเป็นค่าซึ่งสามารถส่งไปใช้ในนิพจน์(expression)อื่นๆได้ การดำเนินการเพื่อกำหนดค่าจึงเป็นเรื่องที่เราจำเป็นต้องกำหนดให้ตัวแปร หรือนิพจน์ที่สนใจอยู่ทางด้านซ้ายมือของเครื่องหมายเท่ากับ-เสมอ ลองดูตัวอย่างต่อไปนี้

นิพจน์	ปฏิบัติการ (operation)	ค่าของนิพจน์
i=3	ใส่มูลค่าเท่ากับ 3 ใน i	3
i=3+4	ใส่มูลค่าเท่ากับ 7 ใน i	7
i=(k=4)	ใส่มูลค่าเท่ากับ 4 ใน k แล้วใส่ ผลลัพธ์ลงใน i	4
i=(k=4)+3	ใส่มูลค่าเท่ากับ 4 ใน k แล้ว ดำเนินการบวก 4 กับ 3 ได้ 7 ใส่ 7 ใน i	7

การปฏิบัติการลักษณะนี้เรียกว่า left hand value (l value) เพราะค่าทางขวาจะถูกใส่ลงในแอดเดรส (address) ที่อยู่ทางซ้าย

นอกจากนี้เรายังมีตัวดำเนินกำหนดค่ารูปอื่น ๆ อีกโดยจะใช้รูปแบบเป็น op= (เดิมใช้เป็น =op ปัจจุบันใช้ op= แทน) โดยที่ op หมายถึง +, -, *, /, %, <<, >>, &, ^, | ดังนั้นนิพจน์ที่เสนอเป็น f op=g; กับนิพจน์ f = f op g ; จะมี

ความหมายเดียวกันเช่น (เราเรียกวิธีนี้ว่า short hand assignment)

```
    a+= 2;   กับ a = a + 2;
  ret* = d;  กับ ret = ret* d;
innum + = 1; กับ innum = innum + 1;
```

หรือ

```
x + = 1;  กับ x = x + 1 ;
x - = 1 ;  กับ x = x - l ;
x * = 2;  กับ x = x * 2 ;
x / = 2 ;  กับ x = x / 2 ;
x % = 2;  กับ x = x % 2; (ก็คือ x mod 2 )
x>>= 1; กับ x = x>> 1; (เลื่อนไปทางขวา 1 บิท)
x<<= 1;  กับ x = x<< 1; ; (เลื่อนไปทางซ้าย 1 บิท)
x & = Ox7f;กับx = x&ox7f; (bitwise AND)
x | = ox7f;กับx = x ox7f ; (bitwise OR)
x A = ox7f;กับx= x^ox7f ; (bitwise XOR)
```

เป็นต้น ตัวดำเนินการกำหนดค่ารูปย่อคือ f op=g หรือ f=f op g นี้เป็นรูปที่เรามีใช้เสมอ ซึ่งจะกล่าวถึงเรื่องที่พาดพิงถึงเรื่องนี้อีกครั้งในบทที่ 5

6) **perfix/postfix operator (+ + , - -)**

perfix operator และ postfix operator ทำหน้าที่เพิ่มค่า (incriment ใช้เครื่องหมาย + +) หรือลดค่า (decrement ใช้เครื่องหมาย - -) ของตัวแปรแล้วแต่กรณีขึ้นอยู่กับการนำเครื่องหมาย + + และ - - ไปวางไว้ด้านหน้าหรือด้านหลังตัวแปร เช่น - - i หมายถึง pre-decrement i + + i หมายถึง pre-increment i i - - หมายถึง post-decrement i i + + หมายถึง post increment i ขอให้สังเกตการทำงานของตัวดำเนินการทั้งสองดังนี้ โดยสมมุติ

ว่า i มีค่าเท่ากับ 5

นิพจน์	ค่าของ i ขณะ operate	ค่าของนิพจน์	ค่าของ i ภายหลัง operation
5+i++	5	10	6
5+i--	5	10	4
--i+5	4	9	4
++i+5	6	11	6

จะเห็นได้ว่ากรณี postfix เราจะดำเนินการภายหลังจากดำเนินการเกี่ยวกับค่าของตัวแปรเสียก่อน เช่น 5+i++ เราใส่มูลค่าเท่ากับ 5 ให้แก่ i ทำให้นิพจน์ 5+i มีค่าเท่ากับ 10 จากนั้นจึงเพิ่มค่าของ i พบว่าภายหลัง operate แล้ว i มีค่าเป็น 6 ใน 5+i-- เราจะใส่มูลค่าเท่ากับ 5 ให้กับ i ทำให้นิพจน์ 5+i มีค่าเท่ากับ 10 จากนั้นจึงลดค่าของ i พบว่าภายหลังจากเมื่อดำเนินการแล้ว i จะมีค่าลดจากเดิมเท่ากับ 5 เหลือเป็น 4 ซึ่งชี้ให้เห็นว่า postfix นั้นจะดำเนินการภายหลังจากที่ตัวแปรถูกนำไปใช้งานแล้ว ขณะที่ prefix จะดำเนินการก่อนที่ตัวแปรจะถูกนำค่าไปใช้ เช่น --i+5 ค่าของ i จะลดจาก 5 เหลือ 4 ก่อนแล้วใส่มูลค่าเท่ากับ 4 ใน i ทำให้นิพจน์ i+5 มีค่าเท่ากับ 9 ++i+5 ค่าของ i จะเพิ่มจาก 5 เป็น 6 ก่อน แล้วจึงใส่มูลค่าเท่ากับ 6 ใน i ทำให้นิพจน์ i+5 มีค่าเท่ากับ 11

กรณีที่ตัวแปร เป็นตัวแปรแบบอื่นก็ปฏิบัติเช่นเดียวกัน ตามตัวอย่างข้างต้นนี้ i เป็น integer variable อนึ่งตัวดำเนินการที่ operate ตัวแปรหรือนิพจน์เราเรียกอีกอย่างหนึ่งว่า infix

7) ตัวดำเนินการเงื่อนไข (conditional operator หรือ tertiary operator,?:)

ตัวดำเนินการเงื่อนไข จะทำงานตามเงื่อนไข หมายความว่าเราจะกำหนดค่าให้แก่นิพจน์ที่สนใจตามเงื่อนไข โดยมีรูปไวยากรณ์ ดังนี้ ขอให้สังเกตว่าต้องมีนิพจน์เกี่ยวข้องอยู่ถึง 3 นิพจน์

```
exp 1 ? exp 2 : exp 3
```

ซึ่งมีความหมายว่านิพจน์ที่ 1 (exp หมายถึง expression 1) เป็นจริง (nonzero) หรือไม่ ถ้าจริงนิพจน์ที่ 1 จะมีค่าเป็นนิพจน์ที่ 2 ถ้าไม่จริงนิพจน์ที่ 1 จะมีค่าเป็นนิพจน์ที่ 3 เช่น

นิพจน์	ความหมาย	มูลค่า
5? 1:2	5 มีค่าเป็น 0 ใช่หรือไม่ถ้าใช่ให้ค่าเป็น 1 ถ้าไม่ใช่ให้ค่าเป็น 2	1
j? i+j:k+j	j มีค่าเป็น 0 หรือไม่ ถ้าเป็นให้ j เท่ากับ i+j ถ้าไม่เป็นให้ j เท่ากับ k+j	
(m>7)? 3:4	m จะมีค่าเท่ากับ 3 ถ้า m มากกว่า 7 และมีค่าเท่ากับ 4 ถ้า m ไม่มากกว่า 7	
(a>b)? a:b	เปรียบเทียบค่าค่าที่มากกว่า ให้เป็น a ค่าที่น้อยกว่าให้เป็น b	
(a>b)? ((a>c)? a:c):((b>c)? b:c)	เปรียบเทียบค่าสูงที่สุดระหว่าง a,b และ c	

หมายเหตุ ที่จริงแล้วตัวดำเนินการเงื่อนไขก็มีความหมายเช่นเดียวกันกับ IF-THEN-ELSE

8) Comma operator (,)

comma operator มิได้ใช้เพื่อเชื่อมนิพจน์เข้าด้วยกัน โดยปกติจะใช้ใน while statement และ for statement การประเมินค่าจะเริ่มประเมินจากนิพจน์ซ้ายสุดเรื่อยไปทางขวาจนถึงนิพจน์ขวาสุด และถือเอามูลค่าของนิพจน์ขวาสุด (แต่ละนิพจน์คั่นด้วยเครื่องหมายจุลภาค คือ ,) เช่น

นิพจน์	มูลค่า
5, 6	6
i++, j+2	j+2
i++, j++, k++	ค่าของ k ก่อนเพิ่มค่า
i++, j++, ++k	ค่าของ k หลังเพิ่มค่า

การเปรียบเทียบตัวดำเนินการในภาษา C กับภาษาอื่น ๆ ปรากฏดังตารางต่อไปนี้ ขอให้สังเกตว่าตัวดำเนินการแบบ <<, >>, ++, --, ?: และ , ไม่มีใช้ในภาษาอื่นหากภาษาอื่นมีใช้ก็ใช้สัญลักษณ์ที่แปลกไปจากนี้

C	BASIC	FORTRAN	PASCAL	PL/I	COBOL
+	+	+	+	+	
					-
*	*	*	*	*	*
/	/	/	/	/	/
-(unary)					
=	=	=	:=	=	=
>	>	GT.	>	>	>
<	<	LT.	<	<	<
>=	>=	.GE.	>=	>=	NOT <
<=	<=	.LE.	<=	<=	NOT >
==	=	.EQ.	=	=	=
!=	<>	.NE.	<>	¬	NOT =
&&	AND	.AND.	AND	&	AND
||	OR	.OR.	OR	|	OR
!	NOT	.NOT.	NOT	1	NOT
%	MOD	MOD ( )	MOD	MOD( )	
&				&	
|				|	
^				BOOL ( )	
~				¬	

The following operators have no direct correspondence in the other languages:

<<
>>
++

?:

2.5 การเปลี่ยนลักษณะตัวแปร (Conversion)

ในกรณีที่เรานำตัวแปรต่างชนิดกันมา operate กันเครื่องจะเปลี่ยน - (convert) ตัวแปรในนิพจน์นั้น ๆ ให้เป็นตัวแปรแบบเดียวกัน โดยอัตโนมัติ โดยทั่วไปแล้ว character variable และ character constant จะถูกเปลี่ยนรูปเป็นจำนวน

เต็มเสมอในทุกนิพจน์เพราะหมายเอาค่าที่เป็นตัวเลขตามรหัสแอสกี้ (ASCII) เรื่องของการเปลี่ยนลักษณะ ตัวแปรนี้เกี่ยวข้องอยู่กับเรื่องของ sign extension เสมอ-เพราะตัวแปรแต่ละแบบใช้พื้นที่ไม่เท่ากัน (ดูตอน 2.1) กฎการเปลี่ยนรูปตัวแปรปรากฏดังนี้

1) ในการทำงานเพื่อประเมินค่าของนิพจน์ใด ๆ ตัวแปรจะมีการเปลี่ยนรูปไปโดยอัตโน-มัติดังนี้คือ

ชนิดของตัวแปร	เปลี่ยนเป็น	หมายเหตุ
char หรือ short	int	sign extension ขึ้นอยู่กับการออกแบบเครื่อง
float	double	float operation ทุกชนิดทำงานในลักษณะ double precision เสมอ

2) ในการทำงานทางคณิตศาสตร์ตัวแปรในลำดับต่ำกว่า หมายถึง มีพิสัยแคบกว่า (ดู-ตอน 2.1) จะเปลี่ยนรูปไปเป็นตัวแปรที่อยู่ในลำดับสูงกว่า (หมายถึงมีพิสัยสูงกว่า) ซึ่งเราจะพบว่าการเปลี่ยนรูปตัวแปรที่อยู่ในลำดับต่ำกว่าไปเป็นตัวแปรที่มีลำดับสูงกว่านั้น จำเป็นจะต้องขยาย sign bit ของตัวแปรเดิม (ที่เปลี่ยนลักษณะ) เพื่อให้จำนวน bit รวมทั้งหมดเท่ากับจำนวน bit ของตัวแปรลำดับสูงกว่า ซึ่งจะมีผลให้เกิด bitwise operation ได้ เช่น int เปลี่ยนลักษณะเป็น long int จะต้องขยาย sign bit อีก 16 บิท หรือ unsigned int เปลี่ยนลักษณะเป็น long int จะต้องขยาย sign bit ที่เป็น 0 (0 หมายถึงเครื่องหมาย +) อีก 16 บิท ดังนี้เป็นต้น

ชนิดของตัวแปร	เปลี่ยนเป็น
int หรือ short int(2 ไบท์)	unsigned int หรือ long int หรือ float หรือ double
unsigned int (2 ไบท์)	long int หรือ float หรือ double
long int (4 ไบท์)	float หรือ double
f l o a t (4 ไบท์)	double
double (8 ไบท์)	

3) ในกรณีตัวดำเนินการกำหนดค่านั้นนิพจน์ทางขวามือจะเปลี่ยนให้มาเป็นชนิดตัวแปรที่อยู่ด้านซ้ายมือ ดังนี้

ชนิดของนิพจน์	ตัวแปรด้านซ้ายมือ	การเปลี่ยนลักษณะ
double	float	ปัดเศษ
float	int	ตัดทศนิยมทิ้งบางส่วน
long	int	ตัดบิทลำดับสูงทิ้ง (ตัด MSB)
int	char	ตัดบิทลำดับสูงทิ้ง (ตัด MSB)

อย่างไรก็ตาม เราสามารถเลือกเปลี่ยนลักษณะตัวแปรให้เป็นแบบใด ๆ ตามความประสงค์ของเราได้โดยอาศัยวิธีการที่เรียกว่า cast construct ซึ่งมีรูปไวยากรณ์ดังนี้คือ

```
(type) exp
```

type หมายถึงชนิดของตัวแปรหรือ data type exp หมายถึงนิพจน์ (expression) เช่น จาก float f = 2.5 (หมายความเดิมเรากำหนดให้ f เป็น float variable มีค่าเริ่มต้นเท่ากับ 2.5) หากประสงค์จะเปลี่ยนรูป f ในเป็น integer variable ก็

กระทำได้ดังนี้คือ (int) f ผลลัพธ์ของ (int) f คือ 2

2.6 ลำดับก่อนหลังของปฏิบัติการ

ตัวดำเนินการในตอน 2.5 ที่ผ่านมานี้โดยปกติจะมีลำดับก่อนหลังในการปฏิบัติการ ยกเว้นมีวงเล็บก็จะทำนิพจน์ในวงเล็บก่อน (ทั้งในแต่ละตัวดำเนินการก็มีลำดับความข้องเกี่ยวกันอยู่เป็นการเฉพาะ) กล่าวคือ ตัวดำเนินการที่มีลำดับสูงกว่าจะทำงานก่อน แต่ถ้ามีลำดับเดียวกันการทำงานจะทำจากซ้ายไปขวา ซึ่งได้สรุปไว้ในตารางต่อไปนี้ จากตารางขอให้สังเกตเส้นแบ่งซึ่งจะแบ่งตัวดำเนินการออกเป็นกลุ่ม ๆ ซึ่งตัวดำเนินการต่าง ๆ ในกลุ่มเดียวกันจะมีลำดับเดียวกัน (equal precedence) กลุ่มที่จัดเรียงไว้ข้างบน จะมีลำดับ (precedance)สูงกว่ากลุ่มที่จัดเรียงไว้ข้างล่าง (ดูตัวอย่าง)

Operator		Associativity	Order of Evaluation
()	function call	left to right	
[]	array element		
->	pointer to structure member		
.	structure member		
!	logical negation	right to left	
~	one's complement		
++	increment		
--	decrement		
-	unary minus		
(*type*)	cast		
*****	pointer		
&	address		
sizeof	size of object		
*	multiplication	left to right	
/	division		
%	modulus		
+	addition	left to right	
-	subtraction		
<<	left shift	left to right	
>>	right shift		
<	less than	left to right	
<=	less than or equal to		
>	greater than		
>=	greater than or equal to		
==	equality	left to right	
!=	inequality		
&	bitwise AND	left to right	
^	bitwise XOR	left to right	
|	bitwise OR	left to right	
&&	logical AND	left to right	left to right
||	logical OR	left to right	left to right
?:	conditional	right to left	
=	assignment	right to left	
***op*=**	assignment		
,	comma	left to right	left to right

ตัวอย่างการทำงานของตัวดำเนินการปรากฏดังนี้

นิพจน์	การทำงาน	คำอธิบาย
x+3*2	x+(3*2)	ทำการคูณก่อน เอา 3 คูณกับ 2 ได้ 6 แล้วผนวก 6 เข้ากับค่าของตัวแปร x
y=x+3*2	y=(x+(3*2))	เอา 3 คูณกับ 2 ได้ 6 แล้วผนวก 6 เข้ากับค่าของ x แล้วใส่ผลลัพธ์ใน y
x=y>>5==7	x=((y>>5)==7)	เคลื่อนค่าของ y ไปทางขวา 5 บิท แล้วเปรียบเทียบค่าดังกล่าวกับ 7 ถ้าเท่ากันให้ใส่ 1 ลงใน x ถ้าไม่เท่ากันให้ใส่ 0 ลงใน x
x=y=z	x=(y=z)	นำค่าของ z ใส่ลงใน y แล้วเอาผลลัพธ์ที่อยู่ใน y ใส่ลงใน x

และในกรณีที่มีตัวดำเนินการผสมกันมากมายหลายชนิดในนิพจน์เดียวกันเราก็ยังคงถือปฏิบัติแบบเดียวกันคือดำเนินการตามลำดับก่อนหลัง (ดูตาราง) เช่น

10<<4/2>> 1 ? 2:6*2+3 && 5|| 2&1|7 สามารถประเมินค่าได้เป็น (ขอให้สังเกตว่าเราจะค่อย ๆ ใส่วงเล็บเพื่อจัดแยกลำดับการทำงาน)

(((10<<(4/2))>>) ? 2 : ((((6*2)+3)&&5)|| ((2&1)| 7)))

2.7 คำสั่ง (statement)

คำสั่งในภาษา C จะต้องลงท้ายด้วยเครื่องหมายอัฒภาคคือ ; เช่น a=5; หรือ j=i+3; หรือ ;(ถ้าคำสั่งใดมีเพียงเฉพาะเครื่องหมายอัฒภาคตามตัวอย่างที่แสดง

ให้ดูนี้เราเรียกว่า null statement) หากเรานำหลาย ๆ คำสั่งมาเขียนต่อกันไว้ภายในวงเล็บปีกกา เราเรียกคำสั่งนั้นว่า คำสั่งเชิงซ้อน (compound statement หรือ block) ซึ่งโดยทั่วไปจะเป็น loop เช่น while loop ดังไวยากรณ์ต่อไปนี้

```
{
คำสั่ง
{
```

เช่น

```
I
i=j ;
j++ ;
{
```

เป็นคำสั่งที่กำหนดค่าของ i ให้เปลี่ยนแปลงไปตามค่าของ j ซึ่งเพิ่มค่าขึ้นเสมอ ในภาษาเบสิคจะใช้เป็น FOR-TO loop ภาษาปาสคาลใช้เป็น BEGIN ... END; ภาษา PL/1 ใช้ BEGIN; ... END; หรือ DO;... END;

อนึ่งภาษา C มีแบบการเขียนที่ไม่เข้มงวดเหมือนภาษาอื่น หมายความว่าเราจะเว้นวรรค เว้นท้ายบรรทัด เว้นย่อหน้ามากน้อยเพียงใดก็ได้ ยกเว้นคำสงวน- (reserve word) คำเฉพาะ (keyword) และ character string เท่านั้นที่ต้องระวังจะเว้นวรรคหรือเขียนติดกันตามอัธยาศัยมิได้ เช่น int i; แปลว่าเรากำหนดให้ตัวแปรชื่อ i เป็น integer variable ขณะที่ inti; หมายถึงตัวแปรชื่อ inti การเปิดทางให้ผู้เขียนคำสั่งมีอิสระในการเว้นวรรคหรือย่อหน้า หรือเว้นระยะ เว้นบรรทัด (เรียกว่า white space) ก็เพื่อให้เกิดความสะดวกสบายในการอ่าน เช่นคำสั่ง

```
{
i=j ;
j++ ;
{
```

ก็สามารถเขียนเป็น { i=j; j++; } ซึ่งเครื่องสามารถรับคำสั่งได้ (จะกล่าวถึงเรื่อง white space อีกครั้งในบทที่ 9)

สำหรับคำอธิบายประกอบคำสั่ง (comment หรือ remark) ให้เขียนในลักษณะ white spaceได้โดยให้เขียนข้อความไว้ภายในเครื่องหมาย /* กับ * / เช่น

```
/* This is subroutine ● /
```

หรือ

```
/*
This
is
 subroutine
*/
```

<u>หมายเหตุ</u> คำสั่งหนึ่ง ๆ อาจมีเพียงเครื่องหมายอัฒภาคคือ ; เท่านั้นก็ได้ซึ่งคำสั่งนี้มีความหมายเช่นเดียวกับคำว่า goto (ดูเรื่อง for ในบทต่อไป)