บทที่ 8
ข้อมูลลักษณะอื่น ๆ

ในบทต่าง ๆ ที่ศึกษาผ่านมานั้นโดยมากจะเป็นการใช้ข้อมูล (data type) 2 แบบคือ int และ char อาจมีบางตัวอย่างที่กำหนดให้ตัวแปรมีข้อมูลเป็น float และ double แม้ว่าเราจะได้ศึกษาถึงข้อมูลหรือตัวแปรแบบต่าง ๆ มาแล้วในบทที่ 1 ในที่นี้จะแสดงตัวอย่างการใช้งานของตัวแปรหรือข้อมูลแบบต่าง ๆ รวมทั้งการผสมข้อมูลต่างแบบเข้าด้วยกันไว้อีกเล็กน้อยให้พอเข้าใจและมองเห็นแนวทางการใช้ประโยชน์

ตัวแปรชนิดต่างๆ(variable type หรือ data type) ที่มีใช้ในภาษา C นั้นหากแจงโดยละเอียดจะปรากฏดังนี้

ชนิดของตัวแปร (หรือข้อมูล)	ขนาด (ไบท์)	พิสัย
char	1	-12.9 ถึง 127 หรือ 0 ถึง 255
int	2	-32768 ถึง 32767
short int	2	-32768 ถึง 32767
long int	4	-214483648 ถึง 214483647'
unsigned int	2	0 ถึง 65536
float	4	-1.7×10^{38} ถึง 1.7×10^{36} (6 digit precision)
long float	8	-1.7×10^{38} ถึง 1.7×10^{38} (16 digit precision)
double	8	-1.7×10^{38} ถึง 1.7×10^{38} (16 digit precision)

8.1 หลักการใช้ตัวแปรผสม (mixing data type)

การใช้ตัวแปรต่างแบบกันในฟังก์ชันเดียวกันนั้น คอมไพรเลอร์จะจัดการแปลงค่าข้อมูลให้สามารถดำเนินการทางคณิตศาสตร์ (mathematical operation) ได้ดังกฎเกณฑ์ต่อไปนี้

กฎที่ 1 ตัวแปรแบบ char จะต้องได้รับการแปลงค่าให้เป็นแบบ int

ตัวอย่างเช่น atoi ( ) ตัวแปร char ใน character string คือ string [i] และ character constant '0' จะถูกเลื่อนชั้น (promote) หรือแปลงให้เป็นตัวแปรแบบ int ก่อนที่จะมีการดำเนินการทางคณิตศาสตร์ เช่น

```
atoi (s)
char  s[ ,1
 {     .
      int i, sign, num ;
          ' ,
        num = 10  * num + s[i]-'0' ;
           :
           :
```

กฎที่ 2 เมื่อมีการกำหนดค่า (assignment) ตัวแปรที่อยู่ทางขวามือของเครื่องหมาย = จะได้รับการแปลงให้เป็นตัวแปรแบบเดียวกันกับตัวแปรที่อยู่ทางซ้ายมือของเครื่องหมายเท่ากับ เช่น

```
number  =  string  111
string [2] = number  +  1 ;
```

string [1] ซึ่งเป็น char จะถูกแปลงเป็น int แล้วใส่ลงในที่ซึ่อ number ซึ่งเป็น int ทำให้มีที่ว่างใน number เหลืออยู่ (char ใช้ที่ 8 บิท number ให้ที่ 16 บิท) จากนั้นจึงใส่ number+1 ลงในที่ซึ่อ string [2] ซึ่งเป็น char กรณีนี้ที่ที่เคยเหลืออยู่ก็มิใช่ปัญหา แต่ถ้าตัวแปรทางขวาใช้ที่มากและถูกจับใส่ (อัด) ลงในที่ที่จุน้อยกว่า ข้อมูลบางส่วนจะถูกตัดออกเพื่อให้พอดีที่ เช่น

```
float big-num ;
int little-num ;
little-num = big-num ;
```

ค่าบางส่วนของ big-num (big-num เป็น float ใช้ที่ 32 บิท) คือค่าหลังจุดทศนิยมจะถูกต้องตัดออกเพื่อให้ใส่ได้พอดีในที่ขนาด 16 บิทของ little-num กรณีนี้เรียกว่ากรณี

ตัดเศษ (truncate)

หมายเหตุ ถ้าตัวแปรใดอยู่ในระดับ float เป็นต้นไปคือ float long float และ double ซึ่งมีค่าทศนิยมด้วยกัน การใส่ค่าแบบ long float หรือ double ลงใน float เราต้องตัดทศนิยมบางตำแหน่งทิ้งบ้าง กรณีนี้เรียกว่าการปัดเศษ (rounding) มิใช่ตัดเศษ

กฎที่ 3 binary operator ต้องใช้ operand 2 ตัวโดย operand ที่ใช้ที่ (จำนวนไบท์) น้อยกว่าจะถูกแปลงให้เป็นตัวแปรหรือข้อมูลแบบเดียวกับ operand ที่ใช้ที่มากกว่าเสียก่อนที่จะมีการดำเนินการทางคณิตศาสตร์ เช่น

```
double num ;
int x, ret ;
    ret = x * num ;
```

กรณีนี้ x เป็น int ใช้ที่ 16 บิทขณะที่ num เป็น double int ใช้ที่ 64 บิท ก่อนที่จะมีการคูณคอมไพเลอร์จะแปลงค่าใน x ให้เป็น double ก่อน เมื่อคูณแล้วจึงใส่ค่าลงในที่ชื่อ ret ซึ่งใส่ได้ไม่หมดเพราะ ret ใช้ที่เพียง 16 บิท กรณีนี้ค่าบางส่วนของ x*num จะถูกตัดทิ้งไปบ้าง (truncate)

สำหรับกรณี ternary operator ซึ่งต้องใช้ operand รวม 3 ตัว ก็ให้ปฏิบัติเช่นเดียวกัน

กฎที่ 4 ถ้านำตัวแปรแบบ float จำนวน 2 ตัว มาดำเนินการทางคณิตศาสตร์ (บวก ลบ คูณ หาร) ตัวแปรทั้งสองจะต้องถูกเลื่อนชั้น (promote) ให้เป็นแบบ double เสียก่อน เมื่อดำเนินการจบจึงใส่ผลลัพธ์ลงในที่ ๆ เตรียมไว้ หากใส่ลงในที่แบบ float จะต้องมีการปัดเศษ (rounding) หากใส่ลงในที่แบบ int จะต้องมีการตัดเศษ (truncate)

กฎที่ 5 ถ้าส่ง char หรือ short เป็นอาร์กิวเมนต์ให้ฟังก์ชัน ทั้ง char และ short จะต้องถูกเลื่อนชั้นเป็น int เสียก่อน และ float ต้องเลื่อนชั้นเป็น double

นอกเหนือจากกฎทั้ง 5 ข้อนี้แล้วเรายังมี cast ซึ่งเป็นวิธีแปลงข้อมูลแบบหนึ่งไปเป็นแบบที่เราต้องการใช้ โดยมากเรามักใช้วิธีนี้ในกรณีที่ต้องการตัดทศนิยมทิ้ง ปัดเศษ หรือปรับค่าให้เหลือขนาดที่พอเหมาะ เช่น ต้องการเอาผลลัพธ์แบบ double ไปเก็บในที่แบบ int ซึ่งโดยปกติจะเก็บไม่พอ และคอมไพเลอร์จะไม่ยอมรับ แต่ถ้าเราต้องการโดยใช้ cast คอมไพเลอร์ก็ไม่ว่าอะไร แต่ถ้าไม่ได้ใช้ cast คอมไพเลอร์จะแจ้ง error messege

รูปแบบไวยากรณ์ของ cast คือ (data type) exp

เช่น result = (int) (dbl-num/int-num)

จากตัวอย่างจะเห็นว่าเราให้ dbl-num เป็น double และ int-num เป็น int ซึ่ง int-num จะต้องเลื่อนชั้นเป็น double เสียก่อนจึงจะไปหาร dbl-num ได้ ผลหารเป็น double ถ้าเรากำหนดให้ที่เก็บชื่อ result เป็น double ก็ไม่มีปัญหาอะไร แต่ถ้า result เป็น int เราจะนำผลหารมาเก็บไม่ได้ ต้องลดชั้นผลหารจาก double

เป็น int ด้วยวิธี cast เสียก่อนจึงจะสามารถเก็บผลในที่ดังกล่าวได้

ต่อไปนี้เป็นตัวอย่างแสดงการเปลี่ยน ASCII characterเป็น floating-point number โดยใช้ชื่อฟังก์ชันเป็น atof ( )

```
/* converts character string to floating-point number*/
double atof (s)
char s [ 1 ;
{
     double val, power ;
     int i, sign ;
     i = 0 ;
     sign = 1;
     if (s [i] = = '+'  || s [i] = = '-')
          sign = (s[i++] = = '+') ? 1:-1 ;
     for (val = 0 ; s[i]> = '0' && s[i] < '9'; i++)
          val = 10 * val + s[i]-'0' ;
     if (s [i] = = '.')
         i + + ;
     for (power = 1 ; s[i] > = '0' && s[i] < = '9' ; i + +) {
          val = 10 * val + s[i] - '0' ;
          power * = 10;
     }
     return (sign * val/power) ;
  }
```

ขณะนี้ฟังก์ชั่น atof ( )รับ character array คือ s [ ] มาจากภายนอกโดย atof ( ) มีภาระที่จะต้องแปลงอะเรย์นี้เป็นตัวเลขโดยส่งคืนในรูป double

ฟังก์ชันกำหนดให้ i เป็น loop counter เริ่มที่ 0 และ sign เป็นที่เก็บเครื่องหมายของ character

ตัวโปรแกรมจะเริ่มต้นด้วยการตรวจสอบว่าสมาชิกตัวแรกของอะเรย์คือ s[0] เป็นเครื่องหมายบวกหรือว่าเครื่องหมายลบอย่างใดอย่างหนึ่งหรือไม่ ถ้าเป็นคำสั่งต่อไปนี้จะถูกเรียกทำงาน (execute)

```
sign = (s[i++] == '+') ? 1:-1 ;
```

คำสั่งนี้ก็คือ

```
if (s[1] == '+')
    sign = 1 ;
else
    sign = -1 ;
```

คำสั่ง `if (s[0] == '+' ||s[i] == '-'` `sign = (s[i++] == '+' ) ? 1 : 1;`

หมายถึงให้ตรวจดูว่าที่ s[0] มีเครื่องหมายบวกหรือลบปรากฏอยู่หรือไม่ แล้วตรวจต่อไปว่าที่ s[1] มีเครื่องหมายบวกหรือว่าลบปรากฏอยู่หรือไม่ ถ้า s[0] เก็บเครื่องหมาย + ให้ใส่ค่าเท่ากับ 1 ลงในที่ชื่อ sign ถ้าเก็บ-ให้ใส่ -1 ลงในที่ชื่อ sign (เหตุที่ต้องเช็คเครื่องหมายทั้งใน s[0] และ s[1] เพราะนี่เป็นเรื่องของ data type ซึ่งเราจะสนใจเฉพาะเครื่องหมายที่อยู่ติดกับตัวเลข เช่น -+1.2 เราจะใส่ใจเฉพาะ +1.2 หรือ +-1.5 เราจะสนใจเฉพาะ -1.5)

จากนั้นจึงเริ่มตรวจ s[2], s[3] เรื่อยไปว่ามีค่าเป็น character ที่อยู่

ระหว่าง 0 ถึง 9 หรือไม่ เมื่อพบแล้วให้แปลงเป็นเลขฐาน 10 (decimal) แล้วเก็บสะสมไว้ในที่ชื่อ val ตามสูตรข้างล่างนี้ โดยทำเช่นนี้เรื่อยไป (ดู i++) จนกว่าจะพบจุดทศนิยมคือ

```
for (val = 0; s[i] > = '0' && s[i] < = '9'; i++)
        val = val * 10 + s[i]-'0' ;
```

เมื่อพบจุดทศนิยมให้ข้ามไปที่ตำแหน่งต่อไป คำสั่งนี้คือ

```
if (s[i] = = '.' )
     i++  ;
```

สมมุติทศนิยมปรากฏที่ s[4] ก็ให้ข้ามไปตรวจที่ s[5] เป็นต้นไปโดยไม่ต้องสะสมค่าจาก s[4] ลงใน val แต่ค่าในที่ตั้งแต่ s[5]เป็นต้นไปในสะสมในval และใน power ดังนี้

```
for(power = 1, s[i]>='0' && s[i]<='9' ; i++) {
     val = 10 * val + s[i] - '0'
     power * = 10
     }
```

ตัวอย่างเช่นเราส่งสตริงชื่อ s ที่มีเลข -128.52 เข้าสู่ฟังก์ชัน atof ( ) เลข 1, 2 8, 5 และ 2 เหล่านี้เป็นรหัสแอสกี้หากกลับเป็น decimal จะมีค่าเป็น 49, 50,56 , 53 และ 50

สมมุติสตริง s ปรากฏดังภาพ

	s[0]	s[1I	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	sign
s	+	-	1	2	8	.	5	2	\0	

1) เมื่อตรวจที่ s[0] พบเครื่องหมายบวก จึงตามไปตรวจ s[1]

2) เมื่อตรวจที่ s[1] พบเครื่องหมายลบ (-) จึงแทนที่ชื่อ sign ด้วย-1

3) ตรวจที่ s[2] พบเลข 1 เห็นว่าเป็นตัวเลขที่มีค่าอยู่ระหว่าง 0 ถึง 9 จริงจึงทำคำสั่ง

```
val = 10 * 0 + 49-48 ;
    = 1 ;
```

4) ตรวจที่ s[3] พบเลข 2 เห็นว่าเป็นตัวเลขที่มีค่าอยู่ระหว่าง 0 ถึง 9 จริงจึงทำคำสั่ง

```
val = 10 * 1 + 50-48 ;
    = 12 ;
```

5) ตรวจที่ s[4] พบเลข 8 เห็นว่าเป็นเลขที่มีค่าอยู่ระหว่าง 0 ถึง 9 จริงจึงทำคำสั่ง

```
val = 10 * 12 + 56-48 ;
    = 128 ;
```

6) ตรวจที่ s[5] พบจุดทศนิยม (.) ให้เลยไปดูที่ s[5]

7) ตรวจที่ s[6] พบเลข 5 เห็นว่าเป็นเลขที่มีค่าอยู่ระหว่าง 0 ถึง 9 จริง จึงทำคำสั่ง

```
val = 10 * 128 + 53-48 ;
    = 1285 ;
```

และ

```
power = 1 * 10
      = 10
```

8) ตรวจที่ s[7] พบเลข 2 เห็นว่าเป็นเลขที่มีค่าอยู่ระหว่าง 0 ถึง 9 จริง จึงทำคำสั่ง

```
val = 1285 * 10 + 50-48 ;
    = 12852 ;
power = 10 * 10 ;
    = 100 ;
```

9) ตรวจที่ s[8] พบ **n u l l** terminater คือ '\0' จึงหยุดตรวจ

ค่าที่ส่งออกจาก atof ( ) ตามคำสั่ง return 'sign*val/power) ;คือ -1 * 12852/100 = -128.52 ซึ่งเป็น double integer(ตามที่กำหนดไว้ว่า double atof (s)

สิ่งที่ควรสังเกตก็คือคำสั่ง

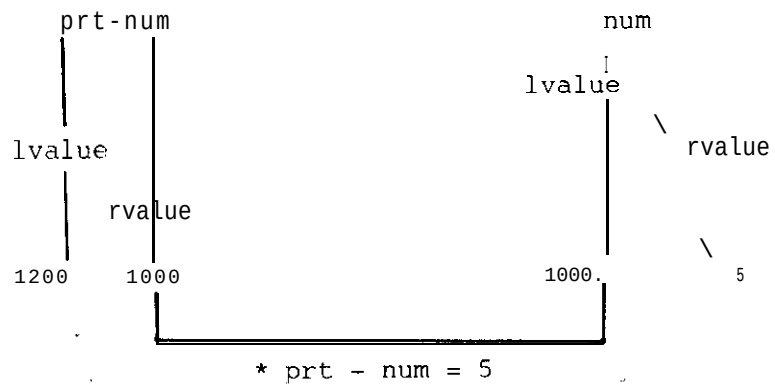
```
val = 10 * val + s[i] - '0' ;
```

จะพบว่าเรากำหนดให้ val เป็น double ขณะที่กำหนดให้ s[i] เป็น char ตามกฎที่ 3 ที่ผ่านมา s[i] จะต้องถูกจัดการเลื่อนชั้น (โดยคอมไพเลอร์) เป็น double เสียก่อน หรือ s[i]-'0' จะต้องถูกเลื่อนชั้น (promote) เป็น double เสียก่อนจึงจะนำไปบวกกับ val * 10 ได้

8.2 การใช้ข้อมูลแบบ Unsign

unsigned int คือข้อมูล (ของตัวแปร) ที่มีค่าเป็นบวกเสมอ และด้วยเหตุที่มีค่าบวกเสมอนี้เองที่ 1 ไบท์ (8 บิท/ไบท์) จึงเก็บตัวเลขได้ตั้งแต่ 0 ถึง 255 ขณะที่ int ต้องใช้ที่ 1 บิทเก็บเครื่องหมาย (0 คือ +, 1คือ -) ที่ 1 ไบท์จึงเก็บตัวเลขได้จาก -128 ถึง + 127 เท่านั้น หากเราใช้ int ในทุกกรณีอาจมีปัญหาที่เก็บไม่พอข้อมูลที่เกินพิสัยจึงอาจสั่งพิมพ์ให้เห็นไม่ได้ เพราะถูกนำไปเก็บในที่ที่เราไม่รู้แอดเดรส

ต่อไปนี้เป็นตัวอย่างการใช้ unsigned data type เพื่อพิมพ์แอดเดรส - (lvalue) และค่า (rvalue) ของตัวแปร ขอให้ระลึกว่า rvalue ของpointerก็คือแอดเดรสของตัวแปรที่ pointer ชี้ไปหา และ lvalue ของ pointer ก็คือแอดเดรสของ pointer ดังภาพ



จากภาพเรากำหนดให้ num เป็น int ตั้งอยู่ ณ แอดเดรส 1000 โดยเก็บค่าเท่ากับ 5 เอาไว้ prt-num เป็น pointer ชี้ไปที่ int ตั้งอยู่ ณ แอดเดรส 1200 ถ้าเรากำหนดให้prt-num = & num; แปลว่าเรากำหนดให้ prt-num มีค่า (rvalue) เท่ากับ 1000 ดังนั้น * prt-num จึงมีค่าเท่ากับค่าที่ปรากฏอยู่ ณ แอดเดรส 1000 คือ 5

โปรแกรมต่อไปนี้อาจช่วยให้เข้าใจเรื่องของ unsigned int และ pointer ดีขึ้นในที่นี้ __ หมาย blank space

```
/* simple program to use unsigned data type */
/* to printout lvalue and rvalue */
/* for a printer and variable */
main ( )
 {
   int num, *prt-num ;
   num = 5 ;
   prt-num = & num ;
   printf ("\n") ;
   printf ( "\t\t prt-num\t\t\t\t num\n" ) ;
   printf ( "\t__ _lvalue ______rvalue\t\t __ _lvalue
                rvalue\n") ,
   printf ("\t\t_ ____\t\t\t\t _____ \\\n") ;
   printf ("\t_____%u_____ %u t t ____%u_____
          %d n", & prt-"urn, prt-num, & "urn, num) ;
   printf ("\t\t\t - - - - - - - - - - - \\ n") ;
   printf ("\t\t\t * prt-"urn = %d", *prt-num) ;

}
```

โปรแกรมนี้กำหนดตัวแปรไว้ 2 ตัวคือ num เป็น int และ prt-num เป็น pointer ชี้ไปที่ int โดยกำหนดค่าไว้ในที่ชื่อ num เท่ากับ 5 แล้วสั่งพิมพ์ lvalue และ rvalue ของตัวแปร หากผู้อ่านยังรู้สึกว่า pointer เป็นเรื่องที่ยากแก่การเข้าใจ ขอให้ทดลองเปลี่ยนค่าแล้วลองวิ่งโปรแกรมดูหลาย ๆ ครั้ง จะเข้าใจได้ดีขึ้น ลักษณะการส่งแอดเดรสรับแอดเดรสและการตามไปรับค่าที่แอดเดรสที่แจ้งมาปรากฏดังภาพที่แสดงให้ดู

แล้วในหน้าก่อน

จากโปรแกรมข้างบน หาก lvalue ของ prt-num คือ 50428 และ lvalue ของ num คือ 50426 ซึ่งมีผลให้ rvalue ของ prt-num เท่ากับ 50426 นั้นจะเห็นว่า int (ใช้ที่ 2 ไบท์) จะรับค่าไม่พอเพราะ int รับค่าได้เพียง -32768 ถึง 32767 เราจึงต้องกำหนดให้เป็น unaigned int ซึ่งรับค่าได้ระหว่าง 0 ถึง 65536 ซึ่งเพียงพอที่จะรับ lvalue และ rvalue ดังกล่าวข้างต้นได้

สำหรับการใช้ long data type นั้นเรามีวัตถุประสงค์เพื่อเพิ่มที่ (จำนวนไบท์) ให้สามารถรับค่าตัวแปรได้มากขึ้น โดยเติมคำว่า long ลงไปด้านหน้า เช่น long int หรือ long float ปกติแล้ว long float จะไม่แตกต่างกับ double และคอมไพเลอร์จะนับเป็นสิ่งเดียวกัน ส่วน long int แม้จะใช้ที่เท่ากับ float แต่กลับใช้เวลาประมวลผลได้รวดเร็วกว่า (ดูบทที่ 1)

8.3 รูปแบบข้อมูลที่ซับซ้อน

การนิยามรูปแบบข้อมูลที่ซับซ้อนนั้นเราอาจมีปัญหาอยู่บ้างเพราะยังไม่คุ้นเคยและไม่ทราบกฎเกณฑ์ อาจทำให้กำหนดผิดหรืออ่านพบแล้วไม่เข้าใจเช่น

```
int (*income [MAX] ( ) ;
```

ซึ่งคงจะงงกันนานพอควรเมื่ออ่านพบครั้งแรก ความจริงแล้วถ้าเราค่อย ๆ พิจารณาก็จะเข้าใจได้ดังนี้

ก่อนอื่นให้เริ่มโดยการดูว่าชื่อตัวแปรคืออะไร เมื่อพบคำว่า income แสดงว่า income คือชื่อตัวแปร จากนั้นให้เริ่มอ่านขวา-ซ้าย ขวา-ซ้าย ของชื่อตัวแปรสลับ

ไปจนจบดังนี้

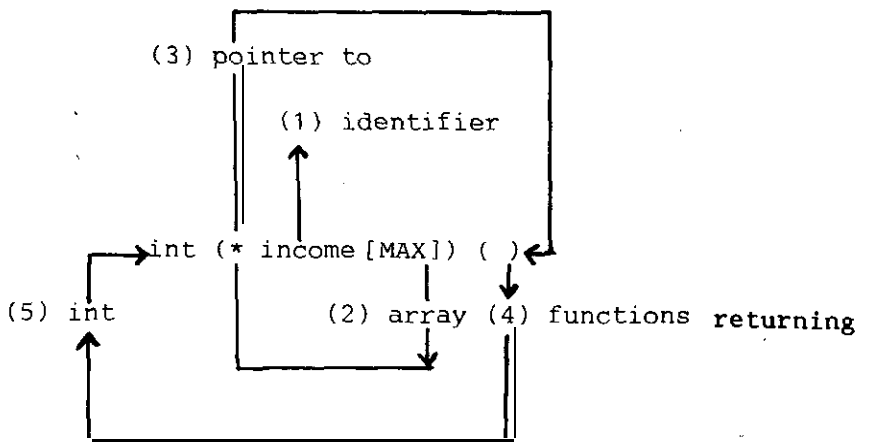
1) ทาง**ขวา**ของ income คือวงเล็บเหลี่ยม [ ] ภายในวงเล็บเหลี่ยมมีคำว่า MAX (เป็น symbolic constant) แสดงว่า income เป็นอะเรย์ขนาดเท่ากับ MAX

2) ทาง**ซ้าย**ของ income [MAX] คือเครื่องหมายดอกจัน (*) แสดงว่า income [MAX] เป็นอะเรย์ของ pointer

3) ทาง**ขวา**ของอะเรย์ของ pointer ชื่อ income คือ (* income-[MAX]) จะพบวงเล็บเล็กคือ ( ) แสดงว่าอะเรย์นี้ชี้ไปที่ฟังก์ชัน

4) ทาง**ซ้าย**ของ (* income [MAX]) ( ) คือ int แสดงว่าฟังก์ชันนั้นส่งค่าเป็น int ไปยังจุดเรียก

กฎเกณฑ์การตรวจแบบขวา-ซ้าย ขวา-ซ้าย ... ของชื่อตัวแปรนี้เรียกว่า right-left rule ซึ่งมีประโยชน์มากในการนิยามของรูปของข้อมูล/ตัวแปรที่ซับซ้อน จากคำอธิบายข้างต้น เราสามารถเขียนเป็นไดอะแกรมได้ดังนี้ ขอให้สังเกตการไหลวนตามลูกศร



อนึ่ง หากเราจำเป็นต้องนิยามข้อมูลที่ซับซ้อนแบบนี้บ่อย ๆ หลาย ๆ ชื่อซึ่งอาจต้องใช้ในโปรแกรมอาจทำให้เกิดข้อผิดพลาดขึ้นได้ และเราคงรู้สึกเป็นภาระมากเกินไปที่ต้องระมัดระวังหรือพะวงกับขวา-ซ้าย ขวา-ซ้าย อยู่บ่อยครั้ง

วิธีหนึ่งที่ทำให้เราสามารถนิยามรูปตัวแปรแบบเดียวกันพร้อม ๆ กันหลาย ๆ ชื่อให้ใช้คอมมานด์ชื่อ typedef (ย่อจากคำว่า type definition) โดยเติมคำว่า typedef ลงหน้ารูปของตัวแปร แล้วกำหนดชื่อกลาง ๆ ขึ้นมาชื่อหนึ่งโดยใช้อักษรตัวพิมพ์ใหญ่ จากนั้นจึงแจงออกมาว่าชื่อกลาง ๆ นั้นรวมเอาตัวแปรชื่ออะไรไว้บ้าง

เช่น เราต้องใช้อะเรย์ของ pointer ที่ชี้ไปที่ฟังก์ชันซึ่งส่งค่าเป็น "int" รวม 3 อะเรย์คืออะเรย์ชื่อ growth, yeild และ heat เราสามารถนิยามรวมไว้ในแห่งเดียวกันได้เป็น

```
typedef int (* PLANT [MAX]) ( ) ;
PLANT growth [MAX], yeild [MAX], heat [MAX] ;
```

ในที่นี้ PLANT เป็นชื่อกลางหรือชื่อร่วม ขอให้สังเกตว่าเราต้องมีเครื่องหมายมหัพภาค-(;) ปิดท้ายคำสั่งเสมอ

หรือในตัวอย่าง
```
typedef char * CHAR-PTR ;
CHAR-PTR messege, prompt ;
```

แสดงว่าเรานิยามให้ messege และ prompt เป็น pointer ชี้ไปที่ char โดยใช้ชื่อร่วมว่า CHAR-PTR กรณีนี้ดูแล้วง่ายกว่าตัวอย่างข้างบน หากนิยามแบบเดิมคือ

```
char * messege ;
char * prompt ;
```

แล้วรู้สึกสบายอกสบายใจกว่าก็ให้ใช้แบบเดิมก็แล้วกันเก็บเอา typedef ไว้ใช้กับรูปแบบ

ข้อมูล/ตัวแปรที่ยุ่งยากซับซ้อนดีกว่า

โดยปกติเรานิยมใช้ typedef สำหรับนิยามโครงสร้างของ file pointer ตัวอย่างการใช้ typedef ในการนิยาม FILE ซึ่งคอมไพเลอร์ใช้ในไฟล์ stdio.h คือ

```
typedef struct
      char * buffer ; /*pointer to file buffer */
      int mode ;      /* file access mode */
      int bufpnt ;    /* index into file buffer */
      int filenum ;   /* file number */
      } FILE ;
```