- The while Statement
- The do while Statement
- Example
- The if Statement
- More on the if Statement

- Characters and Integers
- Floating-Point Variables
- Arithmetic Operators
- Relational and Logical Operators and the Assignment Operator
- Real World Application: Statistical Measures

↓Characters and Integers

The definition of Character

```
char c;
```

creates a 1-byte cell that can hold one character.

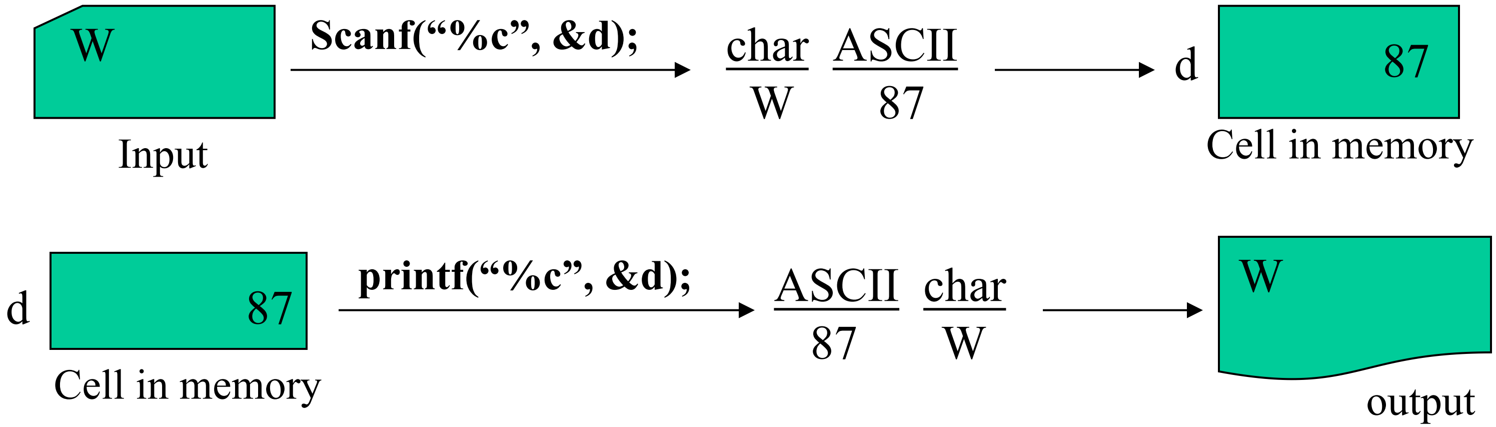
```
c = ‘x’;
```

Note: the character referenced is enclosed in single quotation marks.

```
#include <stdio.h>
main ( ) {
        char c, d;
        c = ‘x’;
        printf ( “Please enter one character: “ );
        scanf ( “%c”, &d );           /* %c-reads the next character in scanf () */
        printf ( “c = %c, ”, c);      /* %c is format descriptor to
                                      print one character  in printf ( ) */
        printf ( “d = %c\n”, d);
}
```

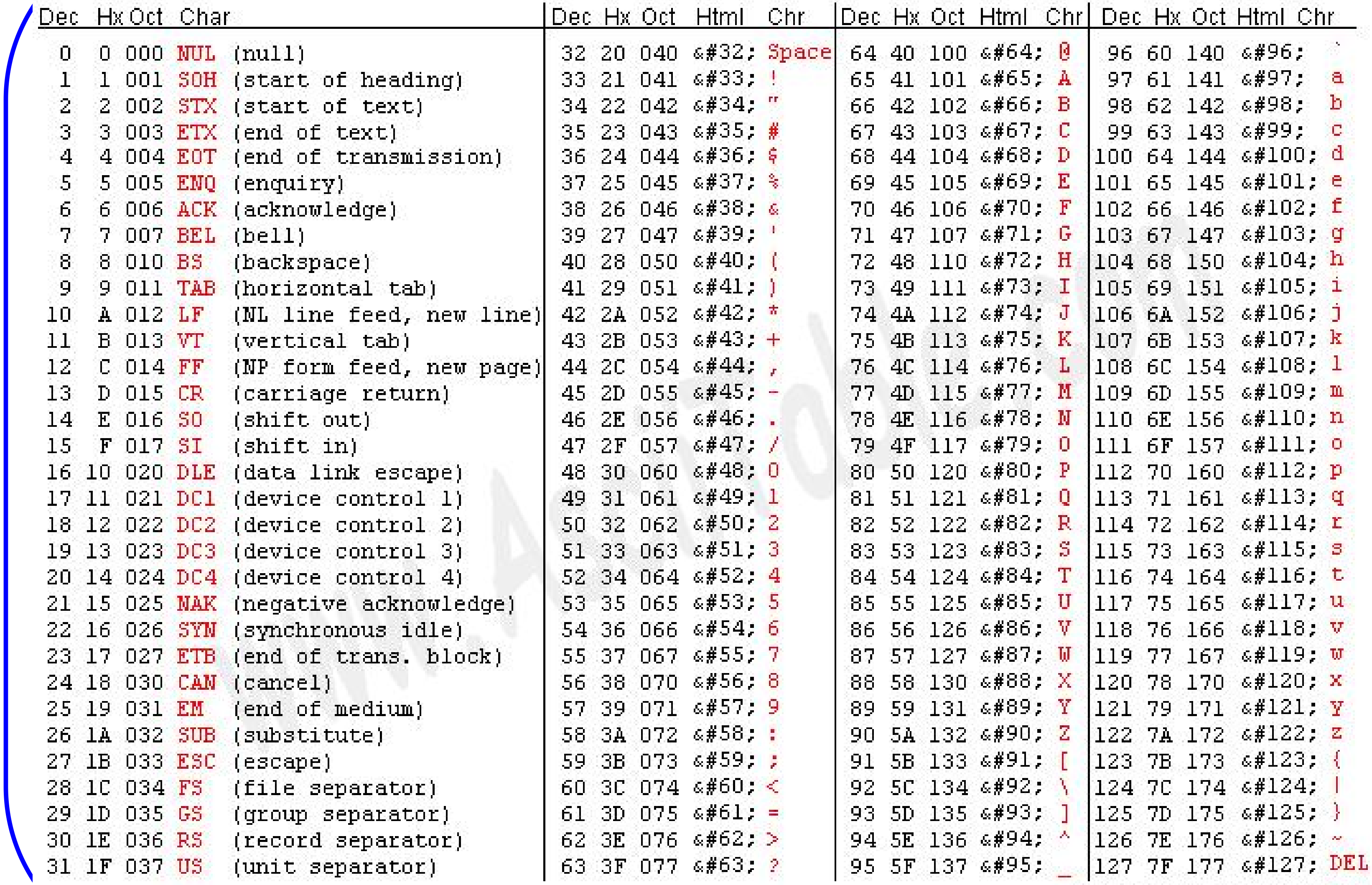
If the input is
W
The output is
c = x, d = W

Reading and writing chars using scanf() and printf( )



W
Scanf(“%c”, &d);
char ASCII
W 87
d 87
Input
Cell in memory

d 87
printf(“%c”, &d);
ASCII char
87 W
W
Cell in memory
output



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

In scanf ( ), %d skip white space until an integer is found. But %c does not skip white space.

```
#include <stdio.h>

main ( ) {

        char c1, c2, c3, c4;

        int x;

        scanf ( “%d%c%c%c%c”, &x, &c1, &c2, &c3, &c4);

        printf ( “%d%c%c%c%c”, x, c1, c2, c3, c4 );

}
```

If the input is
12 345
the output
12 345

If the input is
12 3 45
the output
12 3 4

```
int i;

scanf ( “%c”, &i );      /* Logical Error - for the format descriptor %c, the
          corresponding argument must by the address of          char*/

printf ( “%c”, i );       /* correct */
```

The printf can exchange char and int because system converts char to int before passing the value to printf.

```
#include <stdio.h>
main ( ) {
        char c;
        c = 121;
        printf (“The character of int %d is %c.\n”, c, c );
}
```

The output is (ASCII code)

```
The character of int 121 is y.
```

```
#include <stdio.h>
main ( ) {
        char c;
        c = ‘y’;
        printf (“The character of int %d is %c.\n”, c, c );
}
```

The output is

```
The character of int 121 is y.
```

```
#include <stdio.h>
main ( ) {
        char n;
        int i = 9;
        n = ‘9’;
        printf (“The integer of character %c is %d.\n”, n, n );
        printf (“The character of integer %d is  %c.\n”, i, i );
}
```

The output is

```
The integer of character 9 is 57.
The character of integer 9 is 	.
```

tab

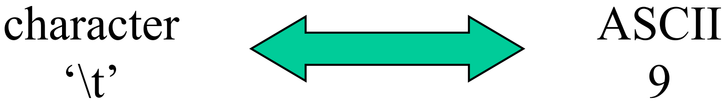
‘\134’;   /*octal digit*/

Represents \ in ASCII code

‘\x5D’   /*hexadecimal digits*/

Represents] in ASCII code

Special Characters



character
'\t'

ASCII
9

'\t' represents one single tab character whose ASCII value is 9.

Codes of special characters.

Constant	Interpretation	ASCII(decimal)	EBCDIC(decimal)
‘\a’	bell	7	47
‘\b’	back space	8	22
‘\f’	form feed	12	12
‘\n’	new line	10	21
‘\t’	tab	9	5
‘\0’	null	0	0
‘\\’	\ (backslash)	92	177
‘\”	‘ (Single quote)	39	125

Interpretation of special characters.

Value of c	Action when we execute printf (“%c”, c);
‘\a’	Bell rings. Printer prints nothing.
‘\b’	Printer moves left one column.
‘\f’	Printer skips to column 1, line 1, of next page.
‘\n’	Printer skips to column 1 of the next line.
‘\t’	Printer skips to next tab position.
‘\0’	Printer prints nothing. (The null character is unprintable.
‘\\’	Printer prints one backslash \.
‘\’’	Printer prints one single quotation mark ‘.

Note: null character ‘\0’ prints nothing, but it uses as a terminator in a data structure such as a string.

```
#include <stdio.h>
main ( ) {
        char c1, c2, c3, c4 , c5, c6;
        c1 = ‘x’;
        c2 = ‘\t’;
        c3 = ‘\\’;
         c4 = ‘y’;
         c5 = ‘\n’;
        c6 = ‘\”;
        printf ( “%c%c%c%c%c%c”, c1, c2, c3, c4 ,c5 ,c6);
}
```

The output is

```
  x        \y
  ’
```

In Nondecreasing order of cell size, integer types are:

char, short int, int, long int ⟷ char, short, int, long

The size of a char cell <= the size of a short int cell, the size of a short int cell <= the size of a int cell and so on.

It is < or = depending on the system.

Signed integer:

signed char, signed short, signed int, signed long

Unsigned binary integer:

unsigned char, unsigned short, unsigned int, unsigned long.

Note: In an arithmetic expression involving integer types, both operands are converted to the first of

int, unsigned int, long, unsigned long

that can represent all values of both types.

Program slice:

```
int i = 2;
char c = ‘y’;
i = i + c;
printf ( “i = %d\n”, i );
```

it will print

```
i = 123     /*y convert to 121 first than plus 2*/
```

- A decimal integer constant - using 0-9, except that the first digit must not be 0.
- An octal integer constant - using 0-7, except that the first digit must be 0.
- A hexadecimal integer constant - using 0-9, a-f, and A-F, preceded by 0x or 0X (the first symbol is zero).

2599, 05047, 0xA27, 0XA27

Decimal, octal and hexadecimal integer

- Decimal
  - 2599
- Octal
  - 05047
  - $5x8^3$+ $0x8^2$+4x8+7=2599
- Hexadecimal
  - 0xA27
  - $10x16^2$+ 2x16+7 =2599
  - A=10, B=11, …, F=15

Range of integer types

Constant	Value on a VAX	Value on an IBM PC
CHAR_MIN	-128	-128
CHAR_MAX	127	127
SHRT_MIN	-32768	-32768
SHRT_MAX	32767	32767
INT_MIN	-2147483648	-32768
INT_MAX	2147483647	32767
LONG_MIN	-2147483648	-2147483648
LONG_MAX	2147483647	2147483647

↓Floating-Point Variables

float,	double,	long double		
↓	↓	↓		
4 byte	8 byte	16 byte	————	Typically

```
float x;            /* define x as float */
double y;           /* define y as double */
x = 7.8;            /* convert 7.8 (dbl) to float 7.8 then store in cell named x */
x = 5;              /* convert integer 5 to the float-point number 5.0 and
                    assigns 5.0 to x*/
y = 8.9;            /* assign number 8.9 to a cell named y */
y = 9;              /* convert 9 to 9.0 and assign 9.0 to y */
x = 123.56e15;      /* assign 123.56 * 10^15      to x */
y = 12.24556E-15;  /* assign 12.24556 *10^-15       to y */
y = x;              /* It is fine. Because double include all of the float values */
```

```
x = y;      /* Some precision will be lost due to truncation */
```

The form descriptor of printf ( )

double and float	%e, %E, %f
long double	%LE, %Le, or %Lf

```
#include <stdio.h>

main ( ) {
            float x, y;
            double z, w;
            x = 12.3456;
            y = 1.2345e-10;
            z = -66.123456;
            w = -88.3456789e16;
            printf ( “x = %e, y = %e\n”, x, y );
            printf ( “x = %E, y = %E\n”, x, y );
            printf ( “x = %f, y = %f\n”, x, y );
            printf ( “z = %e, w = %e\n”, z, w  );
            printf ( “z = %E, w = %E\n”, z, w );
            printf ( “z = %f, w = %f\n”, z, w );
}
```

in VC++, the output is

```
x = 1.234560e+001, y = 1.234500e-010
x = 1.234560E+001, y = 1.234500E-010
x = 12.345600, y = 0.000000
z = -6.612346e+001, w = -8.834568e+017
z = -6.612346E+001, w = -8.834568E+017
z = -66.123460, w = -883456789000000000.000000
```

The form descriptor of scanf ( )

float	%f or %e
double	%lf or %le
long double	%Lf or %Le

```
#include <stdio.h>

main () {
            float x;
            double y;
            scanf ( “%f%lf”, &x, &y );
            printf( “x = %f, y = %f\n”, x, y);
            printf( “x = %e, y = %e\n”, x, y);
}
```

If the input is

```
47.343     -36.30026452e5
```

The output in VC++ is

```
x = 47.342999, y = -3630026.452000

x = 4.734300+001, y = -3.630026e+006
```

Note: Numbers are not exact due to changing between decimal and binary.

↓Arithmetic Operators

+	add
-	subtract
*	multiply
/	divide
%	modulus

```
#include <stdio.h>
main ( ) {
            char c = 8, d = “R”;
            int I = 59, j, k = 2;
            float w = 6.8, x;
            double y = 23.4999e6, z;
            j = I * I ;             /* assign 59 * 59 = 3481 to j */
            j = I * c;              /* the result is int, so assign 59 * 8 = 472 to j */
            j = I / c;              /* the result is int, so assign 7 to j (59 / 8 =
                                    7.375, truncates the fractional part) */
            j = I % k;              /* assign 1 (the remainder of 59 / 2) to j. The
                                    operands I and k must be integers */
```

```
        j= I / w;         /* 59 / 6.8 = 8.676471, the result is integer, so the
                          fractional part is dropped; */

        z = w * y         /* assign  6.8 * 23.4999e6 =1.597993e+8 to z */

}
```

Special kinds of operators that combine arithmetic operations with an assignment

+=, -=, *=, /=, %=

```
x += y + w ;     ⟺     x = x + (y + w);
x -= y - w;      ⟺     x = x - ( y - w);
x /= y * w;      ⟺     x = x / (y * w);
x *= y / w;      ⟺     x = x * (y / w);
x %= y;          ⟺     x = x % y;
```

Precedence of all C operators

Description	Operator	Associates from the	Precedence
Function expr	( )	left	High (Evaluated first)
Array expr	[ ]		
struct indirection	->		
struct member	.		
Incr/decr	++ --	right	
One's complement	-		
Unary not	!		
Address	&		
Dereference	*		
Cast	(type)		
Unary plus	+		
Unary minus	-		
Size in bytes	sizeof		
		left	

left

Mutiplication	*
Division	/
Modulus	%
Addition	+
Subtraction	-
Shift left	<<
Shift right	>>
Less than	<
Less than or equal	<=
Greater than	>
Greater than or equal	>=
Equal	==
Not equal	!=

right

Bitwise and	&	left	
Bitwise exclusive or	^		
Bitwise inclusive or	|		
Logical and	&&		
Logical or	||		
Conditional	? :	right	
Assignment	= %= += -= *= /= >>= <<= &= ^= |=		(Evaluated last)
Comma	,	left	Low

↓Relational and Logical Operators and the Assignment Operator

Examples

- X = 5 + - 3 * 4
  - -7 is stored in x
  - better coding x = 5 + (-3)*4
- y = 5 + 8%3
  - 7
  - because precedence is * / % + -
- to code formula a= b+c, x=1/a:
  - x = 1/(b+c);
  - not x=1/b+c;

Stopped here

Relational operators

==	equal
!=	not equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

Used to compare two values

Logical operators

&&	and
||	or
!	not

The value of an expression involving a relational operator is either true or false.

false is represented by the value 0.

true is represented by any value other than 0.

```
while ( 1 ) {
…
}                          /* while loop executes endlessly because the
                           expression is always true */
while ( -1 ) {
…
}                          /* while loop executes endlessly because the
                           expression is always true */
while ( 0 ) {
…
}                          /* while loop never executes because the expression is
                           always false */
```

Note: When the system evaluates a logical expression, the value assigned is 1 if the expression is true and 0 if the expression is false.

Example

```
while ( 0 ) {
code 1
}
/*
code 1
*/
```

are equivalent if code 1 does not contain any comment

standard C does not allow nested comments:

```
    /*
    …
    /*...
    …*/
    …*/
```

int x = 1, y = 4, z = 14;

Expression	Value	Interpretation
x < y + z	1	True
y == 2 * x + 3	0	False
z <= x + y	0	False
z > x	1	True
x != y	1	True

Note: In both ASCII and EBCDIC, if the character c1 precedes the character c2 in the alphabet, the expression c1 < c2

char c1 = ‘a’, c2 = ‘d’;

c1 < c2 ---- true because ‘a’ precedes the ‘d’ in the alphabet.

char c1 = ‘Y’, c2 = ‘X’;

c1 < c2 ---- false because ‘X’ precedes the ‘Y’ in the alphabet.

The result of using &&, || and ! is as following table.

x	y	x && y	x || y	!x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

exp1 && exp2 means "are both expressions true?"

if exp1 is false, then exp2 is not evaluated

exp1|| exp2 means "is either expression true?"

if exp1 is true, then exp2 is not evaluated

Order of evaluation

! Relation(< = etc.) && ||

int x = 1, y = 4, z = 14;

<table>
<tr><th>Expression</th><th></th><th></th><th>Value</th></tr>
<tr><td>x <= 1 && y ==3</td><td>⟺</td><td>(x <= 1) && (y == 3)</td><td>0</td></tr>
<tr><td>x <= 1 || y == 3</td><td>⟺</td><td>(x <= 1) || (y == 3)</td><td>1</td></tr>
<tr><td>!(x > 1)</td><td></td><td></td><td>1</td></tr>
<tr><td>!x > 1</td><td></td><td></td><td>0</td></tr>
<tr><td>!(x <= 1 || y == 3)</td><td></td><td></td><td>0</td></tr>
<tr><td>x >= 1 && y == 3 || z < 14</td><td>⟺</td><td>(x >= 1 && y == 3) || z < 14</td><td>0</td></tr>
</table>

Note: since x=1 therefore !x=0

Assignment operator =

y=6;

x=y;

assigns y value (6) to x and the expression (x=y) has a value 6

Assignment ‘=‘ associate from right to left

x = y = z; ↔ x = (y = z);

int x = 2, y = 4;

y == (x = 3) + 1; ↔ y == ((x = 3) + 1) equal to 1

The precedence is ‘( )’ ‘+’ ‘==‘

x=1;

x <= 1 && (y = 3) ↔ (x <= 1) && (y = 3) equal to 1

The precedence is ‘( )’ ‘<=‘ ‘&&’

```
int x=2;
if (x=3)
            printf(“true”);
else
            printf(“false”);
```

What will be printed?

What will be printed if (x=3) is changed to (x==3)?

↓Real World Application: Statistical Measures

Problem:

Giving formula: $$\mathbf{Population\_Variance} = \frac{\sum_{i=1}^{n} x_i^2}{n} - \bar{x}_i^2,$$

Where $\mathbf{x_i}$ read from standard input, calculate

1, The count (n)of data items read;

2, The largest value of x;

3, The smallest value of x;

4, $\mathbf{Sum} = \sum_{i=1}^{n} x_i;$

5, $\mathbf{mean} = \frac{\mathbf{Sum}}{n};$

6, $\mathbf{Population\_Variance} = \frac{\sum_{i=1}^{n} x_i^2}{n} - \overline{x_i}^2;$

7, $\mathbf{Standard\_Deviation} = \sqrt{\mathbf{Pupulation\_Variance}}.$

✓ *C Implementation*

```
#include <stdio.h>
#include <math.h>
main ( ) {
        float x,                    /* current value */
        max, min, sum, mean, sum_of_squares,
        variance;                   /* variance of all values read */
        int count = 0;              /* number of values are read so far */
```

```
if (scanf ( “%f”, &x ) == EOF )
    printf ( “0 data items read \n”);
else {
    max = min = sum = x;
    count = 1;
    sum_of_squares = x * x;
    while ( scanf ( “%f”, &x) != EOF ) {
        count += 1;
        if ( x > max )
            max = x;
        if ( x < min )
            min = x;
        sum += x;
        sum_of_squares += x * x;
    } /* while ( scanf ( “%f”, &x) != EOF ) */
    printf ( “%d data items read\n”, count );
    printf ( “maximum value read = %f\n”, max );
    printf ( “minimum value read = %f\n”, min);
```

```
            printf ( “sum of all values read = %f\n”, sum );
            mean = sum / count;
            printf ( “mean = %f\n”, mean );
            variance = sum_of_squares / count - mean * mean;
            printf ( “variance = %f\n”, variance);
            printf ( “standard deviation = %f\n”, sqrt ( variance ) );
    }
}
```

Input

```
2.0 3.0 4.0
```

Output

```
3 data items read
maximum value read = 4.000000
minimum value read = 2.000000
sum of all values read = 9.000000
mean = 3.000000
variance = 0.666667
standard deviation = 0.816497
```