Chapter 3 control flow

Speaker: Lung-Sheng Chien

OutLine

- expression and statement
- selection statement
- iteration statement
- Visual Studio debugger
- goto and labels

What is expression

x = 0	assignment expression
a + b	additive expression
a * b	multiplicative expression
i++	postfix-expression
sizeof(int)	unary expression
(double) x	cast expression
x >> 1	shift expression
1 == x	equality expression
x && y	logical-AND expression
x || y	logical-OR expression
x & y	AND expression
x > y	relational expression
printf(“hello, world”)	postfix expression

Grammar of expression, page 237,238 [1]

expression:
assignment-expression
expression , assignment-expression

assignment-expression:
conditional-expression
unary-expression assignment-operator assignment-expression

assignment-operator; one of
= *= /= %= += -=
<<= >>= &= ^=

conditional-expression :
logical-OR-expression
logical-OR-expression ? Expression : conditional-expression

logical-OR--expression:
logical-AND-expression
logical-OR-expression || logical-AND-expression

logical-AND-expression:
inclusive-OR-expression
logical-AND-expression && inclusive-OR-expression

inclusive-OR-expression:
exclusive-OR-expression
inclusive-OR-expression | exclusive-OR-expression

exclusive-OR-expression:
AND-expression
exclusive-OR-expression ^ AND-expression

Grammar of expression [2]

```
AND-expression:
    equality-expression
    AND-expression & equality-expression
```

```
equality-expression:
    relational-expression
    equality-expression ==  relational-expression
    equality-expression !=  relational-expression
```

```
relational-expression:
    shift-expression
    relational-expression <  shift-expression
    relational-expression >  shift-expression
    relational-expression <=  shift-expression
    relational-expression >=  shift-expression
```

```
shift-expression:
    additive-expression
    shift-expression <<  additive-expression
    shift-expression >>  additive-expression
```

```
additive-expression:
    multiplicative-expression
    additive-expression +  multiplicative-expression
    additive-expression -   multiplicative-expression
```

```
multiplicative-expression:
    cast-expression
    multiplicative-expression *  cast-expression
    multiplicative-expression /  cast-expression
    multiplicative-expression %  cast-expression
```

```
cast-expression:
    unary-expression
    ( type-name )  cast-expression
```

```
unary-expression:
    postfix-expression
    ++  unary-expression
    - -  unary-expression
   unary-operator cast-expression
   sizeof  unary-expression
   sizeof ( type-name )
```

```
unary-operator: one of
    &  *  +  -  ~  !
```

Grammar of expression [3]

```
postfix-expression:
   primary-expression
   postfix-expression  [ expression ]
   postfix-expression  ( argument-expression-list )
   postfix-expression .  Identifier
   postfix-expression ->  identifier
   postfix-expression ++
   postfix-expression - -
```

```
argument-expression-list
   assignment-expression
   argument-expression-list , assignment-expression
```

```
primary-expression
   identifier
   constant
   string
   ( expression )
```

x is identifier

page 192: identifier is a sequence of letters and digits, the first character must be a letter

```
constant
   integer-constant
   character-constant
   floating-constant
   enumeration-constant
```

“hello, world” is string

1234 is integer-constant

3.5E+2 is floating-constant

Example x = 0 [1]

expression:
    assignment-expression
    expression , assignement-expression

assignment-expression x = 0

assignment-expression:
    conditional-expression
    unary-expression assignment-operator assignment-expression

unary-expression = assignment-expression
x = 0

unary-expression:
    postfix-expression
    ++ unary-expression
    - - unary-expression
    unary-operator cast-expression
    sizeof unary-expression
    sizeof (type-name)

postfix-expression x

postfix-expression:
    primary-expression
    postfix-expression [expression]
    postfix-expression (argument-expression-list)
    postfix-expression . Identifier
    postfix-expression -> identifier
    postfix-expression ++
    postfix-expression - -

primary-expression x

Example x = 0   [2]

```
primary-expression
    identifier
    constant
    string
    ( expression )
```

```
assignment-expression:
    conditional-expression
    unary-expression assignment-operator assignment-expression
```

```
conditional-expression :
    logical-OR-expression
    logical-OR-expression ? Expression : conditional-expression
```

```
logical-OR--expression:
    logical-AND-expression
    logical-OR-expression ||  logical-AND-expression
```

```
logical-AND-expression:
    inclusive-OR-expression
    logical-AND-expression && inclusive-OR-expression
```

```
inclusive-OR-expression:
    exclusive-OR-expression
    inclusive-OR-expression |  exclusive-OR-expression
```

identifier	x
conditional-expression	0
logical-OR-expression	0
logical-AND-expression	0
inclusive-OR-expression	0
exclusive-OR-expression	0

Example x = 0   [3]

```
exclusive-OR-expression:
    AND-expression
    exclusive-OR-expression ^  AND-expression
```

```
AND-expression:
    equality-expression
    AND-expression & equality-expression
```

```
equality-expression:
    relational-expression
    equality-expression ==  relational-expression
    equality-expression !=  relational-expression
```

```
relational-expression:
    shift-expression
    relational-expression <  shift-expression
    relational-expression >  shift-expression
    relational-expression <=  shift-expression
    relational-expression >=  shift-expression
```

```
shift-expression:
    additive-expression
    shift-expression <<  additive-expression
    shift-expression >>  additive-expression
```

AND-expression	0
equality-expression	0
relational-expression	0
shift-expression	0
additive-expression	0

Example x = 0 [4]

additive-expression:
  multiplicative-expression
  additive-expression + multiplicative-expression
  additive-expression - multiplicative-expression

multiplicative-expression:
  cast-expression
  multiplicative-expression * cast-expression
  multiplicative-expression / cast-expression
  multiplicative-expression % cast-expression

cast-expression:
  unary-expression
  (type-name) cast-expression

unary-expression:
  postfix-expression
  ++ unary-expression
  - - unary-expression
  unary-operator cast-expression
  sizeof unary-expression
  sizeof (type-name)

multiplicative-expression	0
cast-expression	0
unary-expression	0
postfix-expression	0

Example x = 0   [5]

```
postfix-expression:
    primary-expression
    postfix-expression  [ expression ]
    postfix-expression  ( argument-expression-list )
    postfix-expression .  Identifier
    postfix-expression ->  identifier
    postfix-expression ++
    postfix-expression - -
```

primary-expression   0

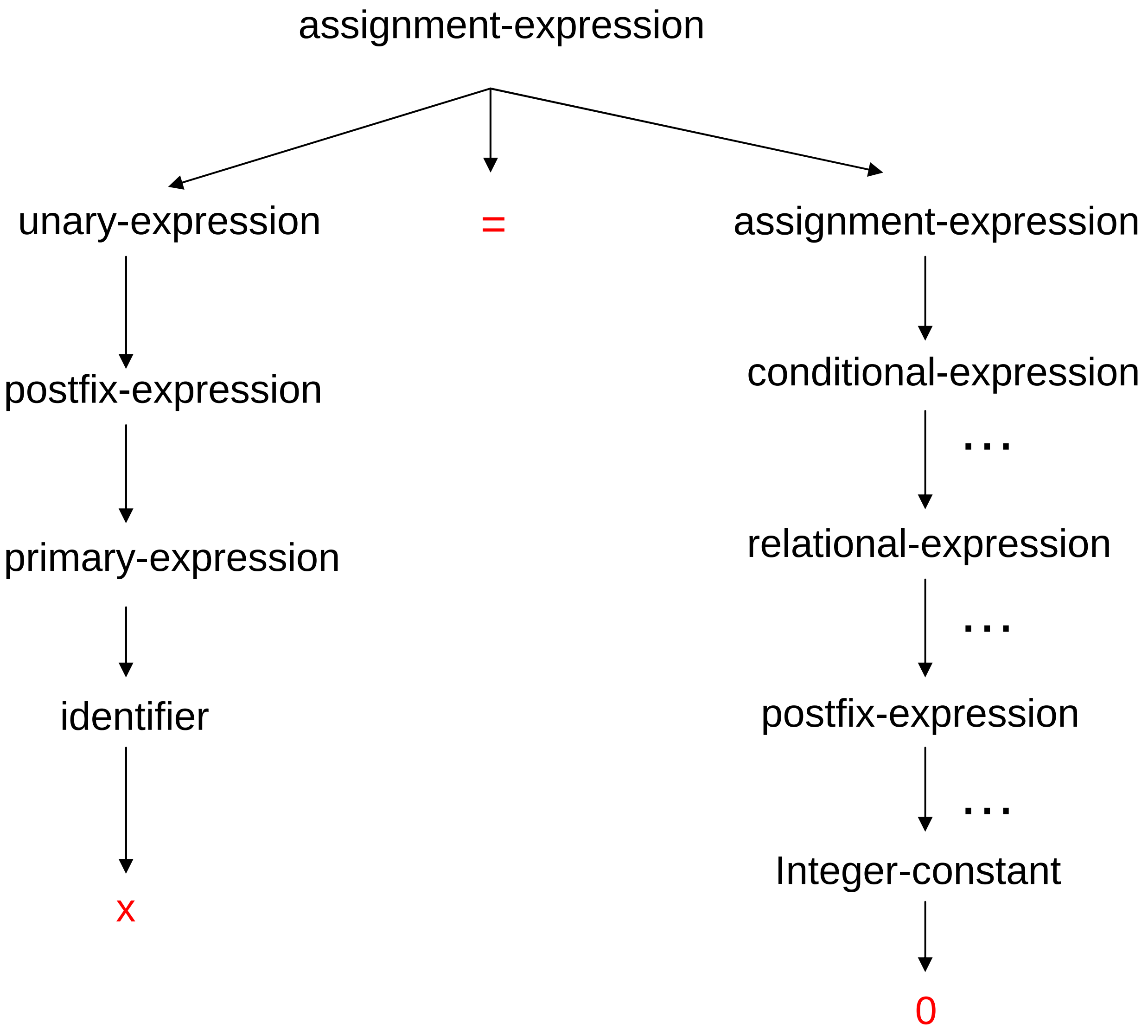
```
primary-expression
    identifier
    constant
    string
    ( expression )
```

constant   0

```
constant
    integer-constant
    character-constant
    floating-constant
    enumeration-constant
```

Integer-constant   0

Parsing tree of expression x = 0



assignment-expression

unary-expression

postfix-expression

primary-expression

identifier

x

=

assignment-expression

conditional-expression

...

relational-expression

...

postfix-expression

...

Integer-constant

0

What is statement

Labeled statement	case 5 : y = 3 ;
Expression statement	x = 0 ;

Selection statement	If (a > b)
	x = a ;
	else
	x = b ;

Compound statement	{
	x = 5 ; y = 3 ;
	}

Iteration statement	for(i = 0 ; i < 5 ; i++){
	a[i] = b[i] + c[i] ;
	}

Jump statement	return 0 ;

Grammar of statement   page 236, 237

```
statement:
   labeled-statement
   expression-statement
   compound-statement
   selection-statement
   iteration-statement
   jump-statement
```

```
labeled-statement:
   identifier : statement
   case constant-expression : statement
   default : statement
```

```
expression-statement:
   expression ;
```

```
compound-statement:
   { declaration-list  statement-list }
```

```
statement-list:
   statement
   statement-list statement
```

```
selection-statement:
   if ( expression ) statement
   if ( expression ) statement  else  statement
   switch (expression) statement
```

```
jump-statement:
   goto  identifier ;
   continue ;
   break ;
   return  expression ;
```

```
iteration-statement:
   while ( expression ) statement
   do statement while ( expression )  ;
   for (expression ; expression ; expression )  statement
```

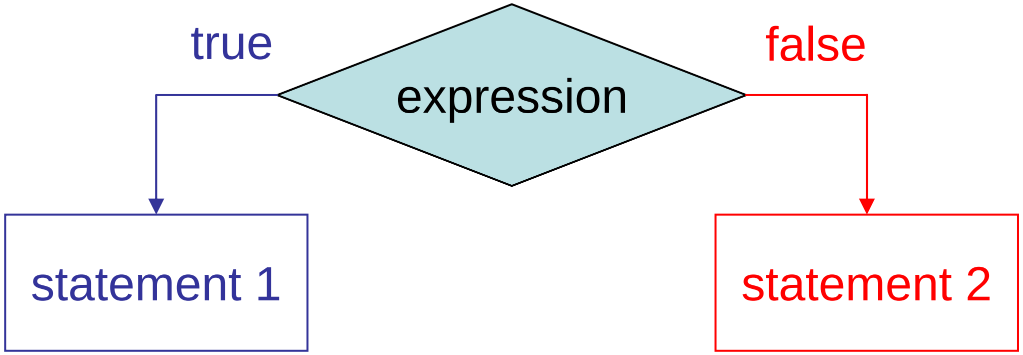
OutLine

- expression and statement
- selection statement
  - if-then-else statement
  - switch-case statement
- iteration statement
- Visual Studio debugger
- goto and labels

If-Else statement (selection-statement) :

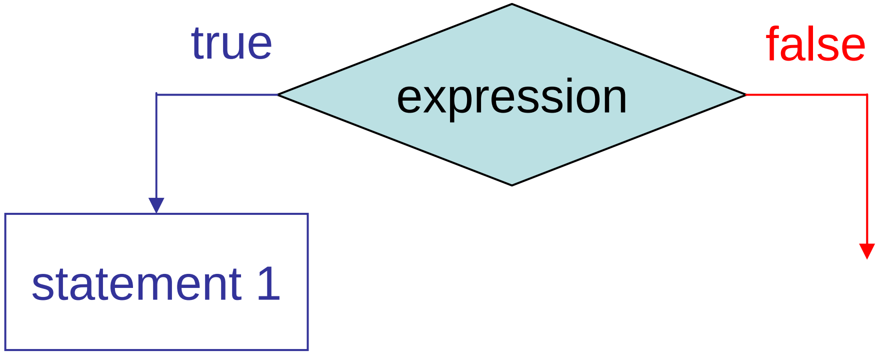
two-way decision

```
If ( expression )
    statement 1
else
    statement 2
```



single-way decision

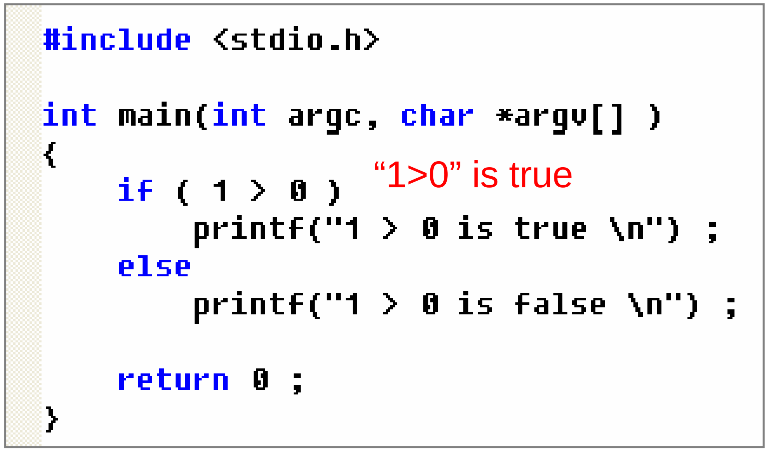
```
If ( expression )
    statement 1
```



Question: How to determine true or false of expression ?

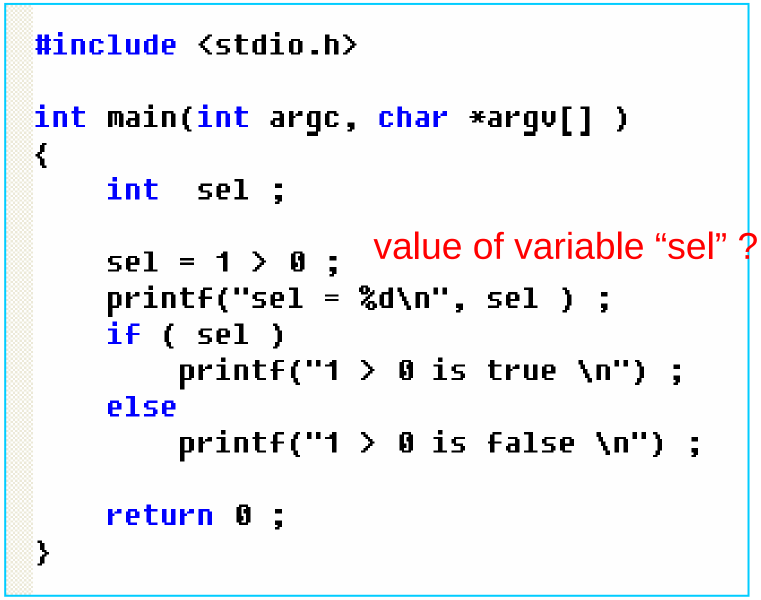
If expression is non-zero, then it is true, otherwise it is false

Example 1 (0 means false)



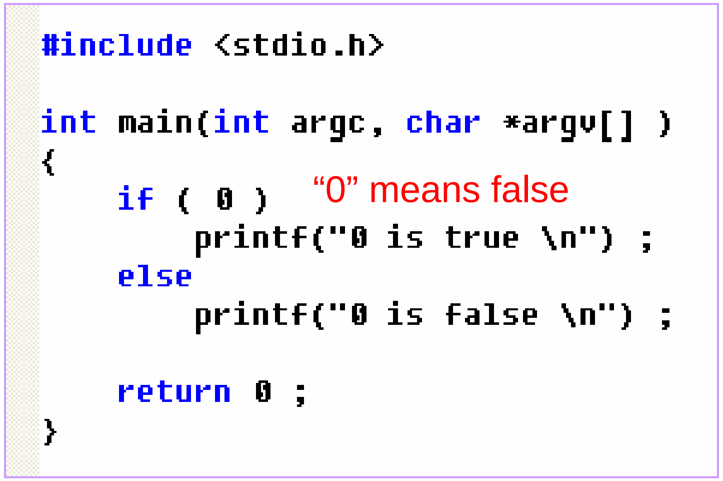
```c
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( 1 > 0 )
        printf("1 > 0 is true \n") ;
    else
        printf("1 > 0 is false \n") ;

    return 0 ;
}
```

“1>0” is true



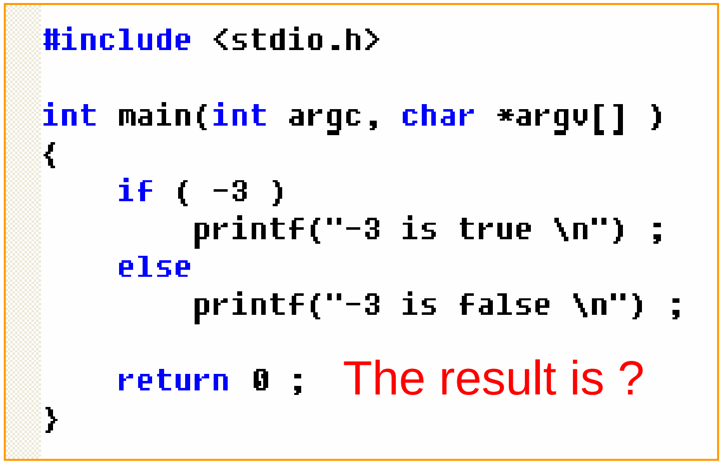
```c
#include <stdio.h>

int main(int argc, char *argv[] )
{
    int  sel ;

    sel = 1 > 0 ;
    printf("sel = %d\n", sel ) ;
    if ( sel )
        printf("1 > 0 is true \n") ;
    else
        printf("1 > 0 is false \n") ;

    return 0 ;
}
```

value of variable “sel” ?



```c
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( 0 )
        printf("0 is true \n") ;
    else
        printf("0 is false \n") ;

    return 0 ;
}
```

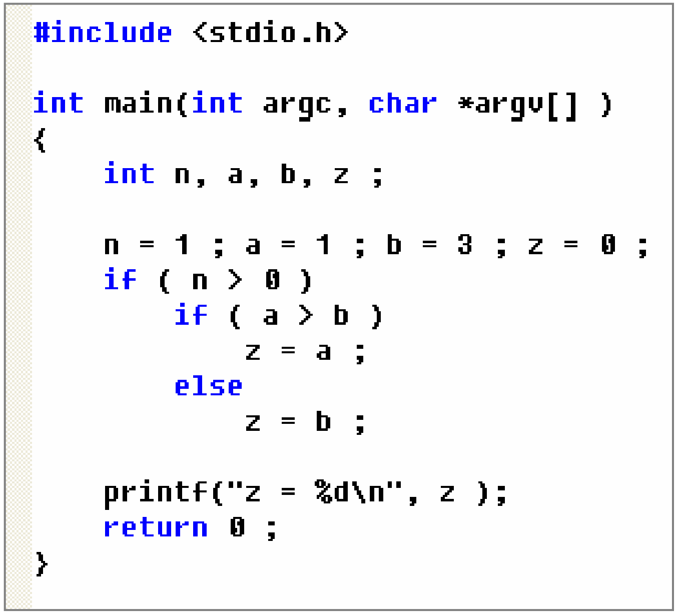
“0” means false



```c
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( -3 )
        printf("-3 is true \n") ;
    else
        printf("-3 is false \n") ;

    return 0 ;
}
```

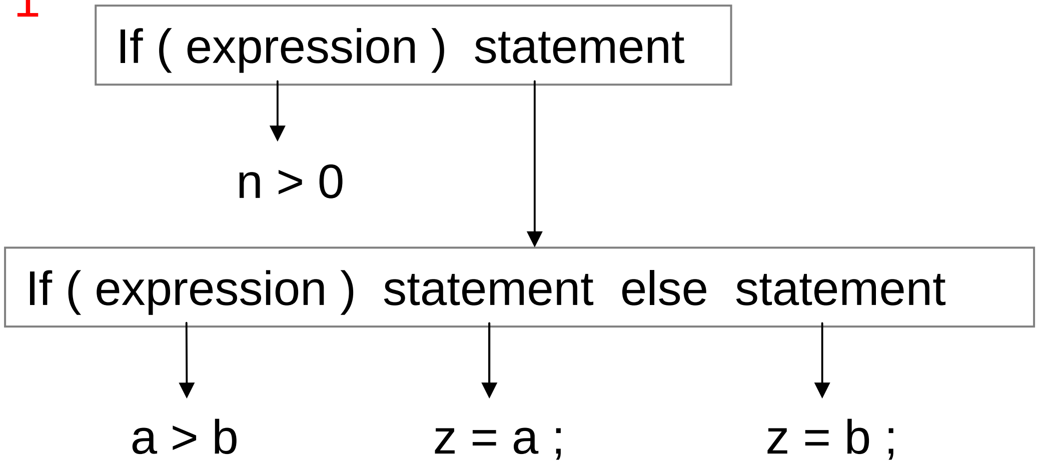
The result is ?

Example 2: ambiguity of nested if-then-else



```
#include <stdio.h>

int main(int argc, char *argv[] )
{
    int n, a, b, z ;

    n = 1 ; a = 1 ; b = 3 ; z = 0 ;
    if ( n > 0 )
        if ( a > b )
            z = a ;
        else
            z = b ;

    printf("z = %d\n", z );
    return 0 ;
}
```

Case 1



If (expression) statement

n > 0

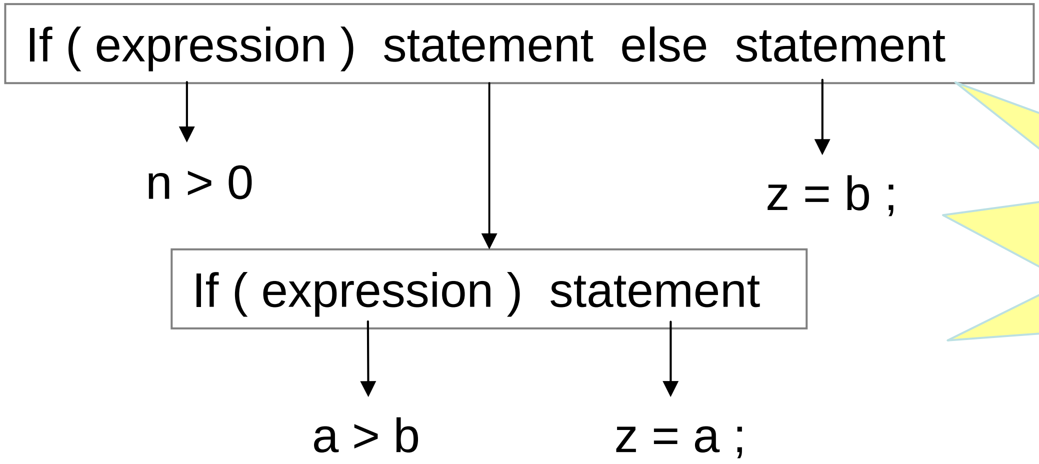
If (expression) statement else statement

a > b

z = a ;

z = b ;

Case 2



If (expression) statement else statement

n > 0

z = b ;

If (expression) statement

a > b

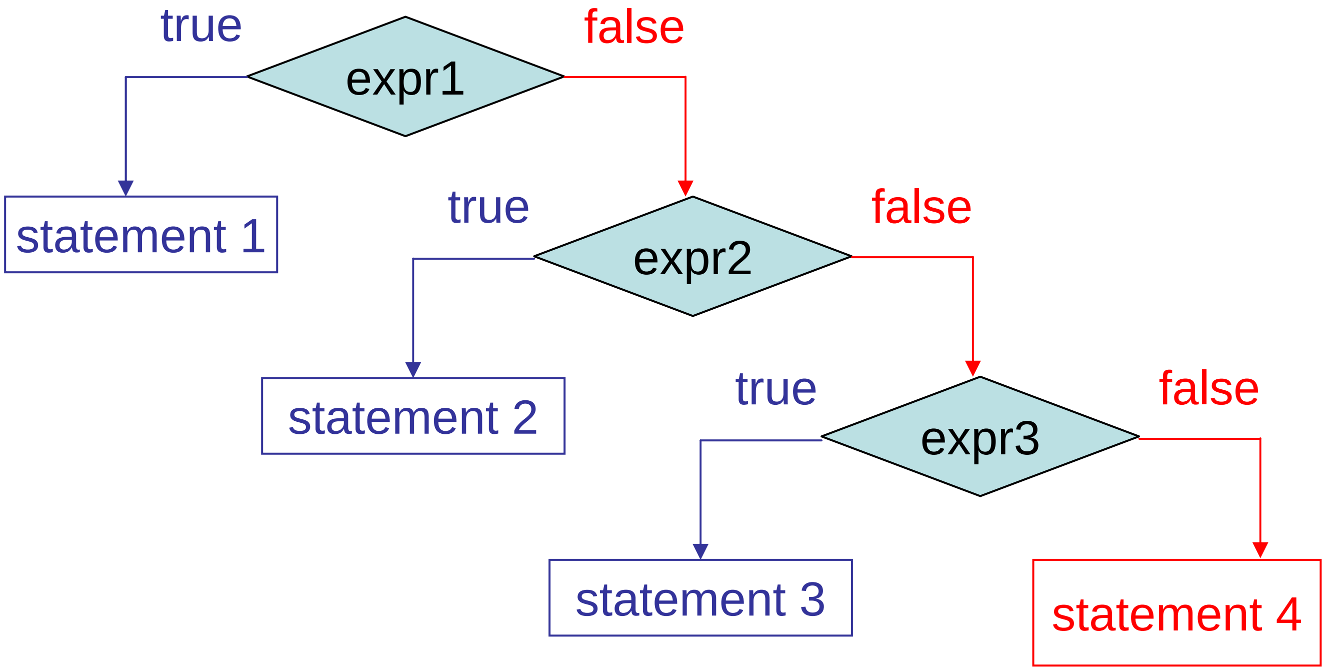
z = a ;

else associates to closest else-less **if**

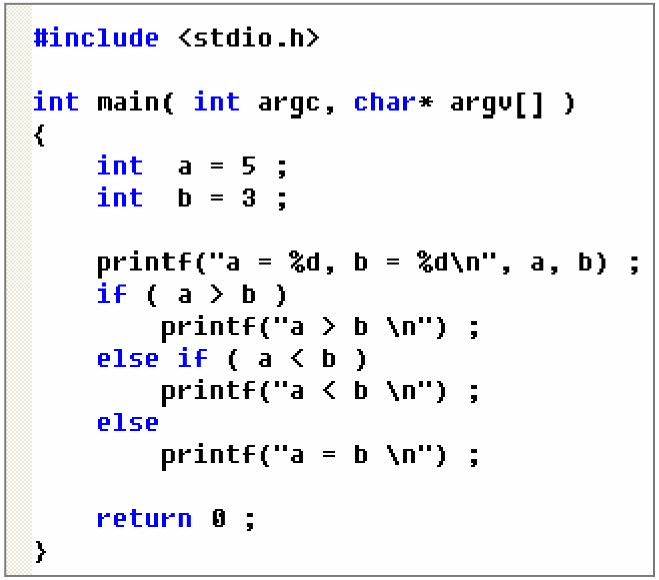
Else-if statement (selection-statement)

multi-way decision

```
If ( expr1 )
    statement1
else if ( expr2 )
    statement2
else if ( expr3 )
    statement3
else
    statement4
```

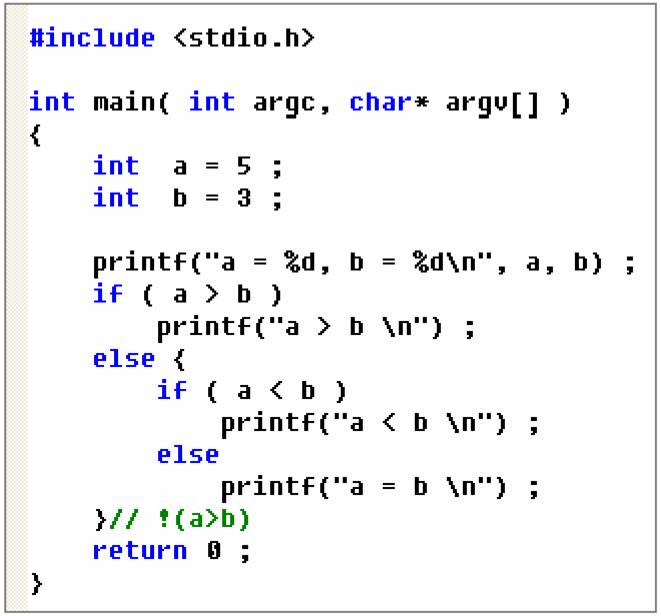


Example 3: trichotomy (三分律)

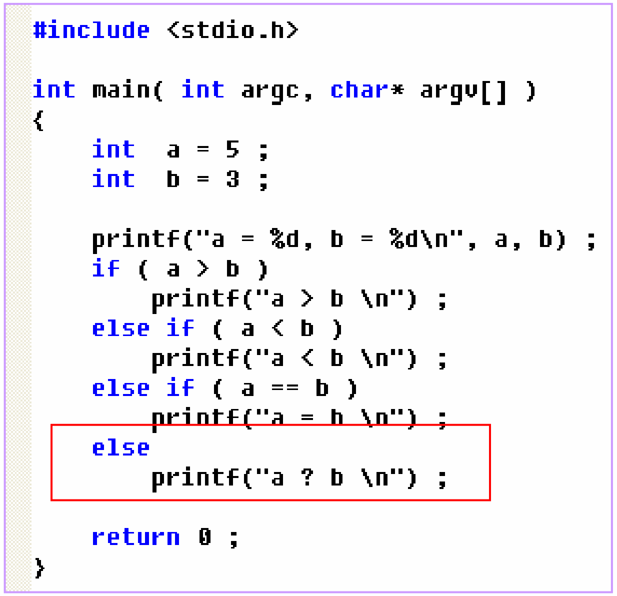


```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int  a = 5 ;
    int  b = 3 ;

    printf("a = %d, b = %d\n", a, b) ;
    if ( a > b )
        printf("a > b \n") ;
    else if ( a < b )
        printf("a < b \n") ;
    else
        printf("a = b \n") ;

    return 0 ;
}
```

=



```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int  a = 5 ;
    int  b = 3 ;

    printf("a = %d, b = %d\n", a, b) ;
    if ( a > b )
        printf("a > b \n") ;
    else {
        if ( a < b )
            printf("a < b \n") ;
        else
            printf("a = b \n") ;
    }// !(a>b)
    return 0 ;
}
```



```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int  a = 5 ;
    int  b = 3 ;

    printf("a = %d, b = %d\n", a, b) ;
    if ( a > b )
        printf("a > b \n") ;
    else if ( a < b )
        printf("a < b \n") ;
    else if ( a == b )
        printf("a = b \n") ;
    else
        printf("a ? b \n") ;

    return 0 ;
}
```

Don’t write such code

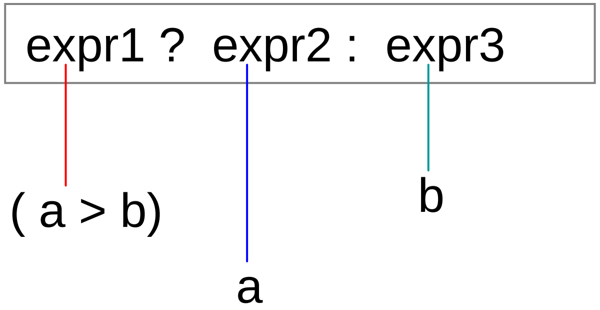
Conditional expression (page 51)

```
If ( a > b )
   z = a ;
else
   z = b ;
```

=

```
z = ( a > b) ? a : b ;
```

?: is called ternary operator

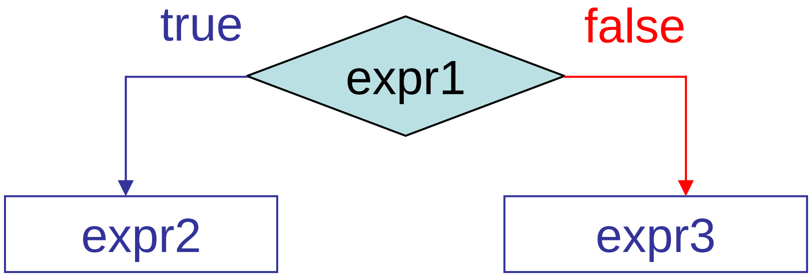


expr1 ? expr2 : expr3

(a > b)

a

b



true

expr1

false

expr2

expr3

switch statement (multi-way decision)

```
switch ( expr ) {
    case const_expr1 : statement1
    case const_expr2 : statement2
    case const_expr3 : statement3
    default : statement4
}
```

=

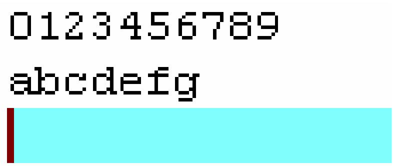
?

```
If ( expr == const_expr1 )
    statement1
else if ( expr == const_expr2 )
    statement2
else if ( expr == const_expr3 )
    statement3
else
    statement4
```

Example 4: count digit, white space, others (in page 59)

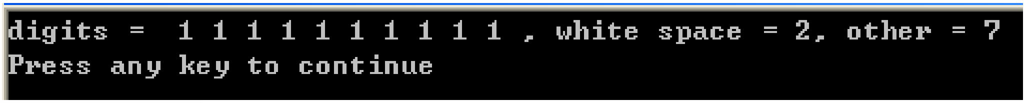
```c
#include <stdio.h>
#include <assert.h>

#define     FILENAME  "data.txt"
#define     NUM_DIGIT   10

int main( int argc, char* argv[] )
{
    int c, i, nwhite, nother, ndigit[NUM_DIGIT] ;
    FILE*  fp ;  // file descriptor

    fp = fopen( FILENAME, "r" ) ;  // open file named FILENAME
    assert( fp ) ;  // verify whether file exists

    nwhite = nother = 0 ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        ndigit[i] = 0 ;

    while( ( c = fgetc(fp) ) != EOF ){
        switch(c) {
        case '0' : case '1' : case '2' : case '3' : case '4' :
        case '5' : case '6' : case '7' : case '8' : case '9' :
            ndigit[c - '0'] ++ ;
            break ;
        case ' ' :  case '\n' :  case '\t' :
            nwhite ++ ;
            break ;
        default:
            nother++ ;
            break ;
        }// switch
    }

    printf( "digits = ") ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        printf(" %d", ndigit[i] );

    printf(" , white space = %d, other = %d\n", nwhite, nother);

    fclose(fp) ;  // close file descriptor

    return 0 ;
}
```

File data.txt



```
0123456789
abcdefg
```

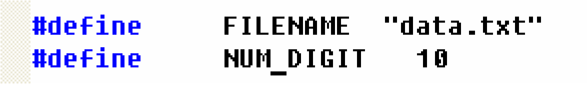
result



```
digits =  1 1 1 1 1 1 1 1 1 1 , white space = 2, other = 7
Press any key to continue
```

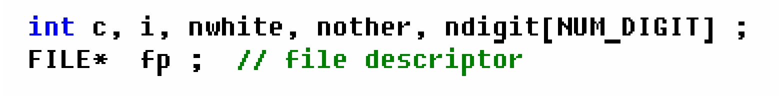
decomposition of example 4

1.Macro definition



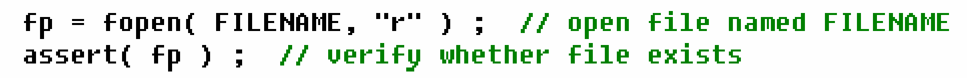
```
#define     FILENAME  "data.txt"
#define     NUM_DIGIT   10
```

2. declaration

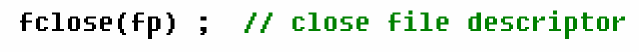


```
int c, i, nwhite, nother, ndigit[NUM_DIGIT] ;
FILE*  fp ;  // file descriptor
```

3. File handle

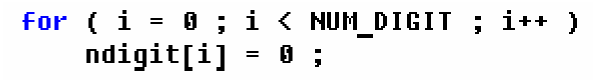


```
fp = fopen( FILENAME, "r" ) ;  // open file named FILENAME
assert( fp ) ;  // verify whether file exists
```

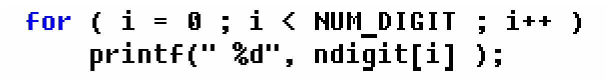


```
fclose(fp) ;  // close file descriptor
```

4. for-loop

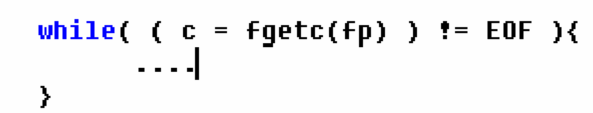


```
for ( i = 0 ; i < NUM_DIGIT ; i++ )
    ndigit[i] = 0 ;
```



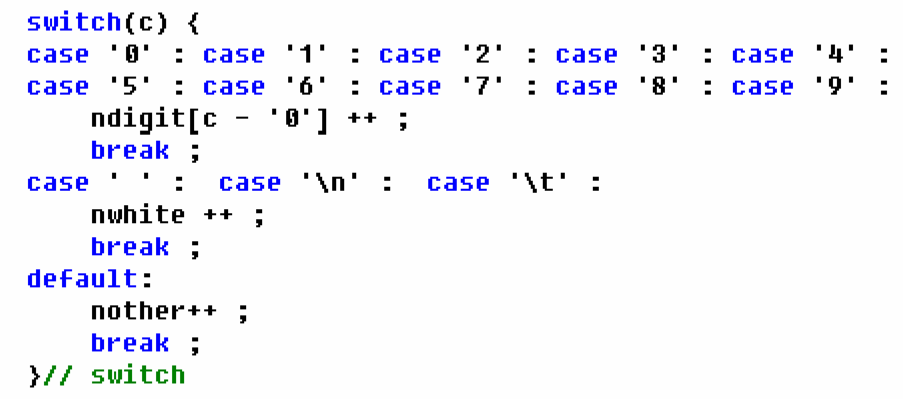
```
for ( i = 0 ; i < NUM_DIGIT ; i++ )
    printf(" %d", ndigit[i] );
```

5. while-loop



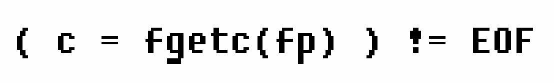
```
while( ( c = fgetc(fp) ) != EOF ){
      ....
}
```

6. switch statement



```
switch(c) {
case '0' : case '1' : case '2' : case '3' : case '4' :
case '5' : case '6' : case '7' : case '8' : case '9' :
    ndigit[c - '0'] ++ ;
    break ;
case ' ' :  case '\n' :  case '\t' :
    nwhite ++ ;
    break ;
default:
    nother++ ;
    break ;
}// switch
```

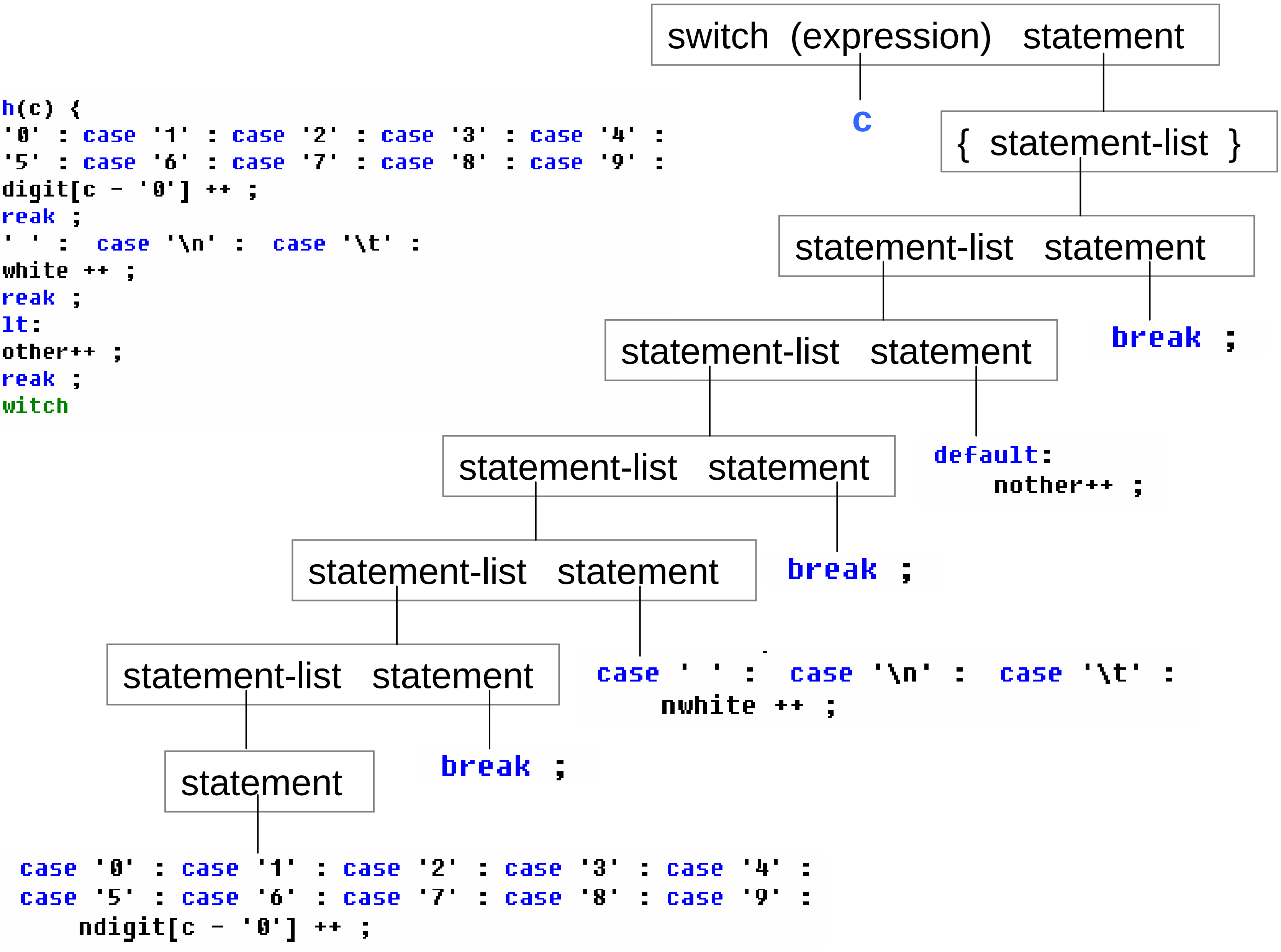
7. weird expression



```
( c = fgetc(fp) ) != EOF
```

grammar of switch statement

```
switch(c) {
case '0' : case '1' : case '2' : case '3' : case '4' :
case '5' : case '6' : case '7' : case '8' : case '9' :
    ndigit[c - '0'] ++ ;
    break ;
case ' ' :  case '\n' :  case '\t' :
    nwhite ++ ;
    break ;
default:
    nother++ ;
    break ;
}// switch
```



switch (expression) statement

c

{ statement-list }

statement-list statement

break ;

statement-list statement

default:
    nother++ ;

statement-list statement

break ;

statement-list statement

case ' ' : case '\n' : case '\t' :
    nwhite ++ ;

statement-list statement

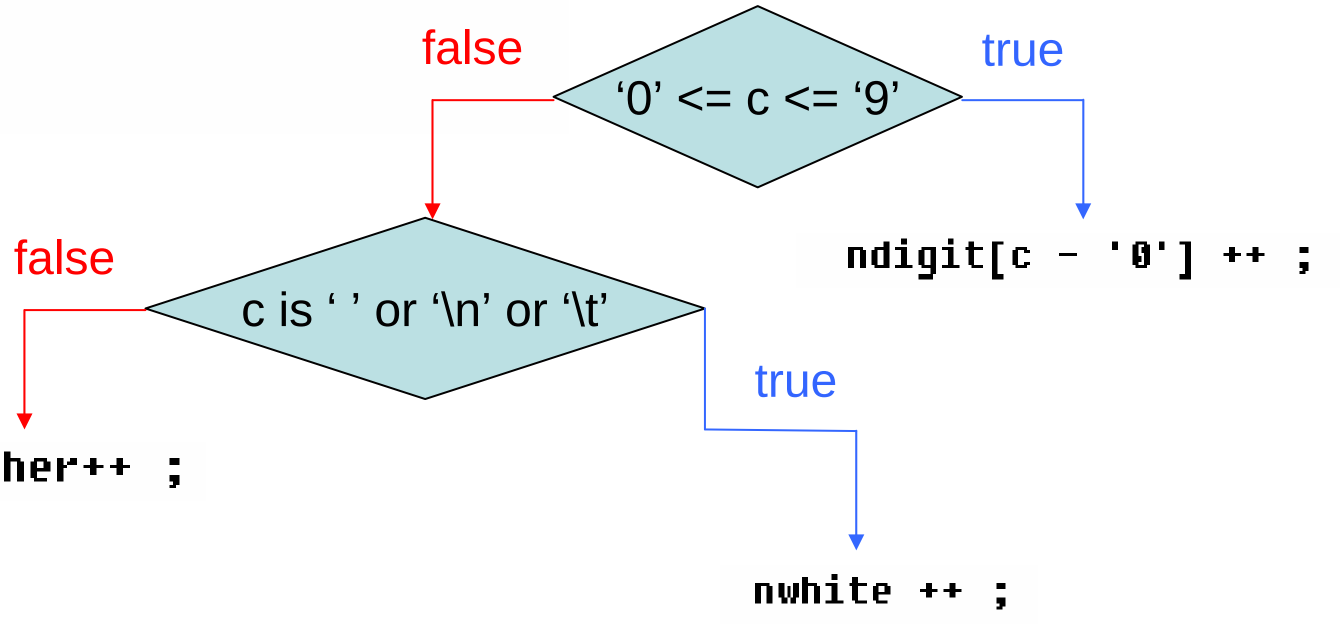
break ;

statement

case '0' : case '1' : case '2' : case '3' : case '4' :
case '5' : case '6' : case '7' : case '8' : case '9' :
    ndigit[c - '0'] ++ ;

decision tree of switch statement

```
switch(c) {
case '0' : case '1' : case '2' : case '3' : case '4' :
case '5' : case '6' : case '7' : case '8' : case '9' :
    ndigit[c - '0'] ++ ;
    break ;
case ' ' :  case '\n' :  case '\t' :
    nwhite ++ ;
    break ;
default:
    nother++ ;
    break ;
}// switch
```



false

‘0’ <= c <= ‘9’

true

false

c is ‘ ’ or ‘\n’ or ‘\t’

true

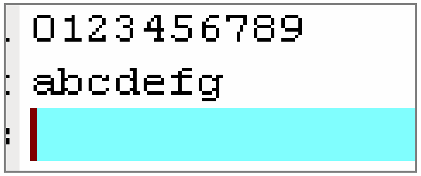
`ndigit[c - '0'] ++ ;`

`nother++ ;`

`nwhite ++ ;`

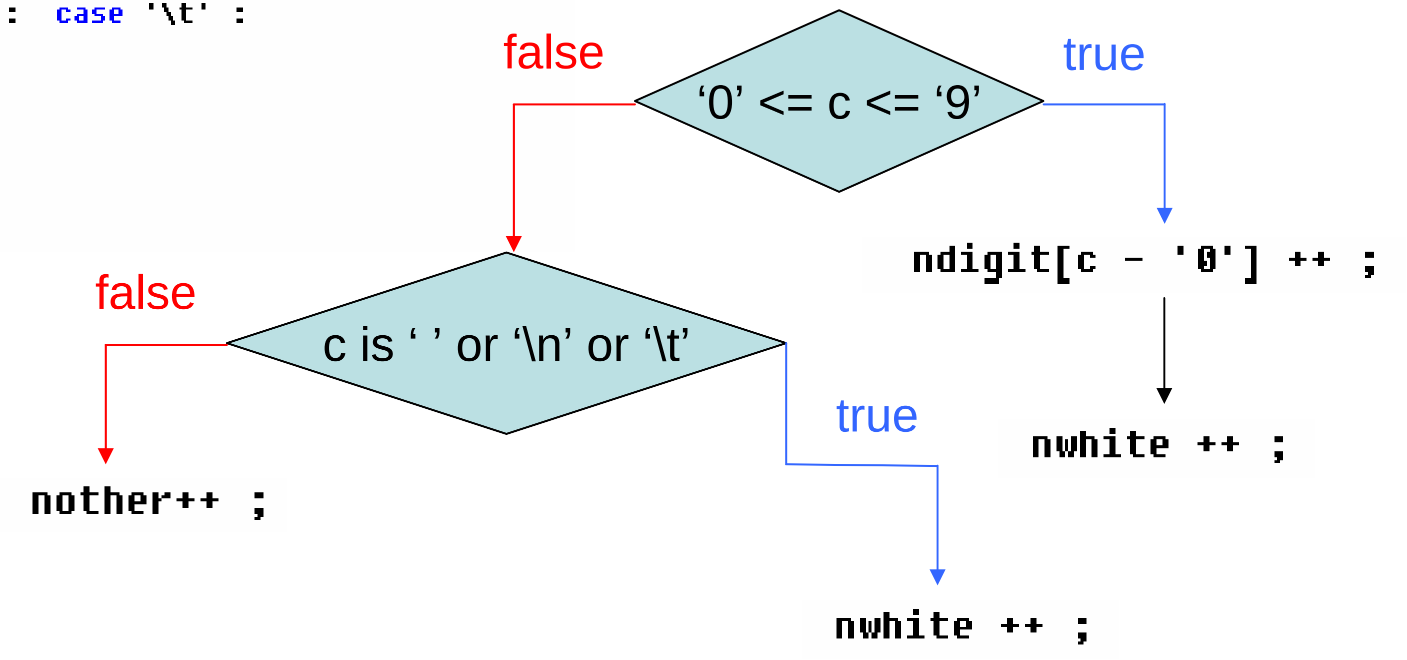
if miss one break statement, then …

File data.txt

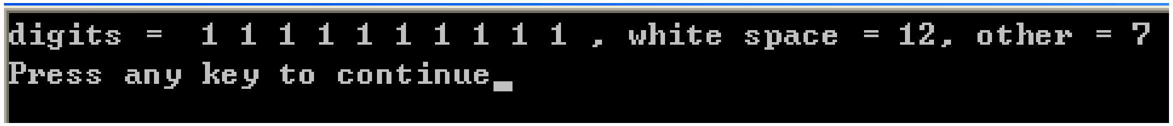


```
0123456789
abcdefg
```

```
switch(c) {
case '0' : case '1' : case '2' : case '3' : case '4' :
case '5' : case '6' : case '7' : case '8' : case '9' :
    ndigit[c - '0'] ++ ;

case ' ' :  case '\n' :  case '\t' :
    nwhite ++ ;
    break ;
default:
    nother++ ;
    break ;
}// switch
```



result

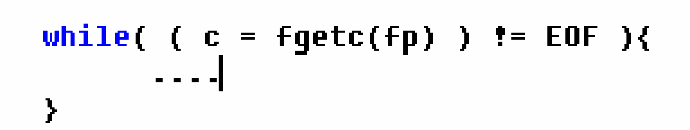


```
digits =  1 1 1 1 1 1 1 1 1 1 , white space = 12, other = 7
Press any key to continue_
```

OutLine

- expression and statement
- selection statement
- iteration statement
  - while-loop
  - for-loop
- Visual Studio debugger
- goto and labels

while-loop

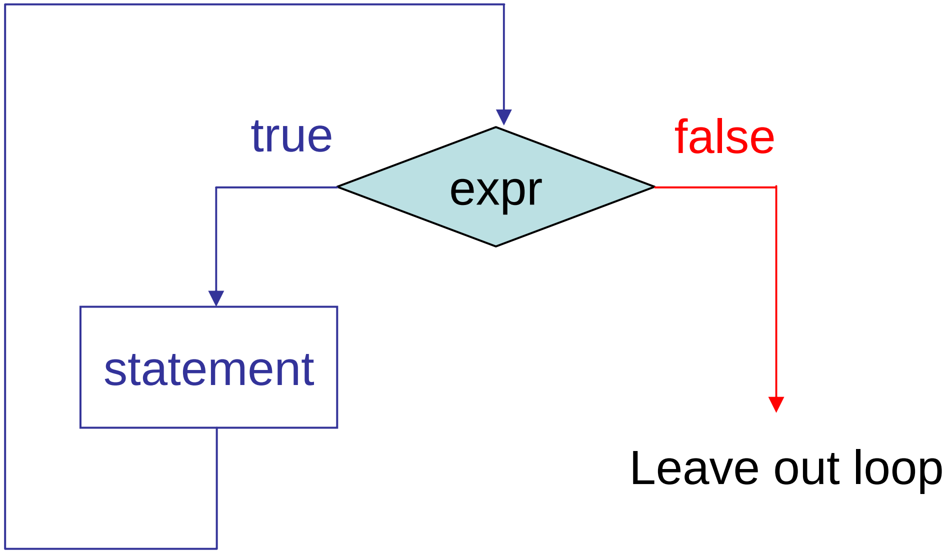


```
while( ( c = fgetc(fp) ) != EOF ){
        ....
}
```

```
while ( expr )
      statement
```

Infinite loop

```
while ( 1 )
      statement
```



for-loop

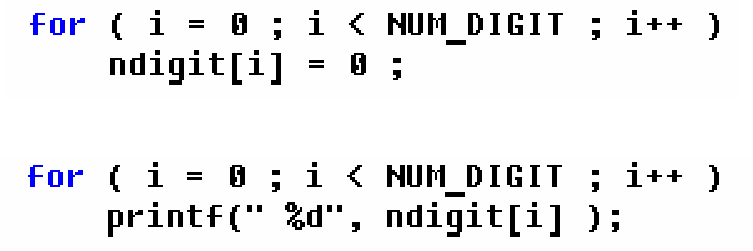
```
for ( expr1 ; expr2 ; expr3 )
      statement
```

=

```
expr1 ;
while ( expr2 ){
      statement
      expr3 ;
}
```

Infinite loop

```
for ( ;  ;  )
      statement
```



```
for ( i = 0 ; i < NUM_DIGIT ; i++ )
    ndigit[i] = 0 ;

for ( i = 0 ; i < NUM_DIGIT ; i++ )
    printf(" %d", ndigit[i] );
```

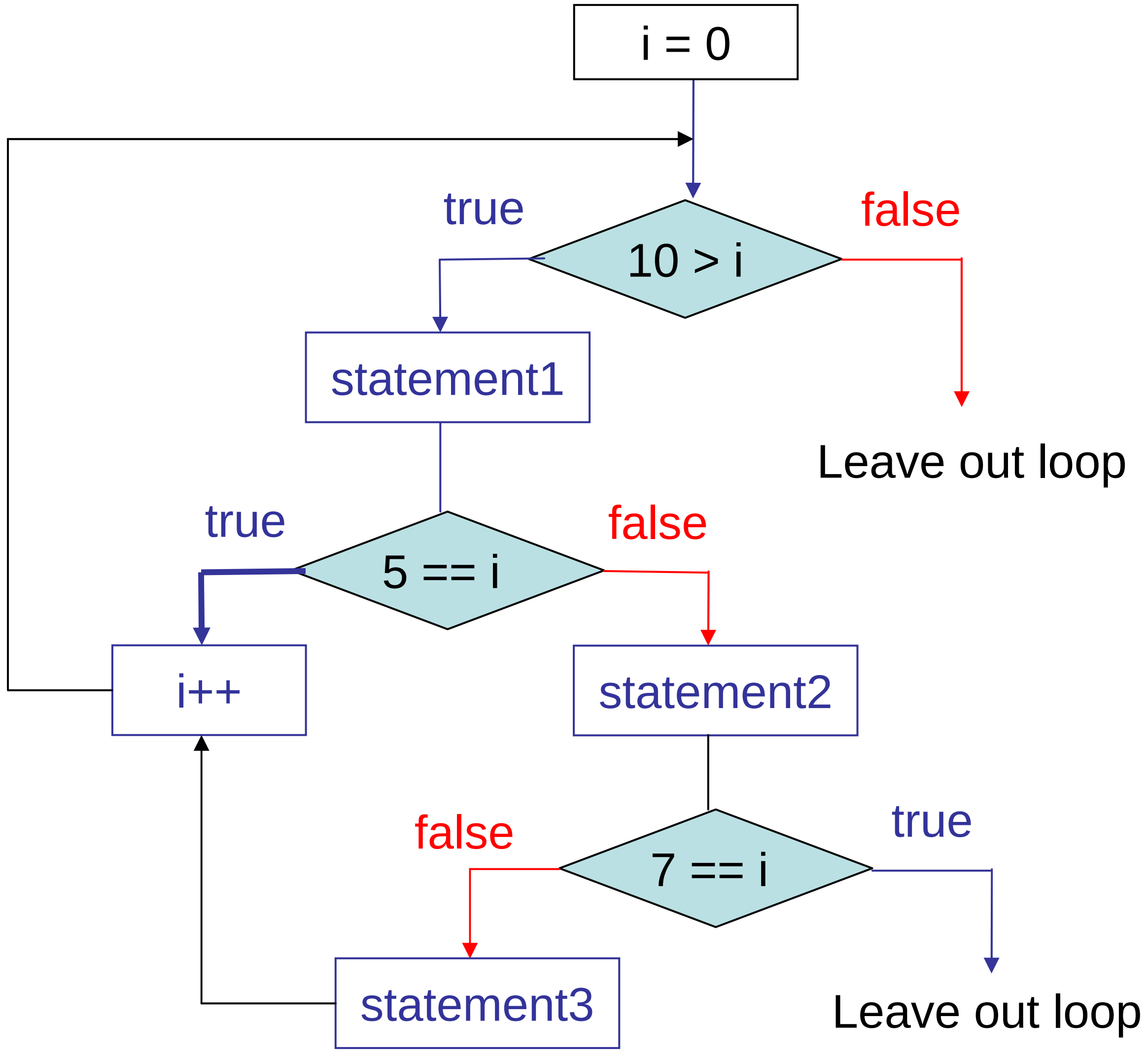
expr1: initial setting

expr2: termination condition

expr3: incremental step

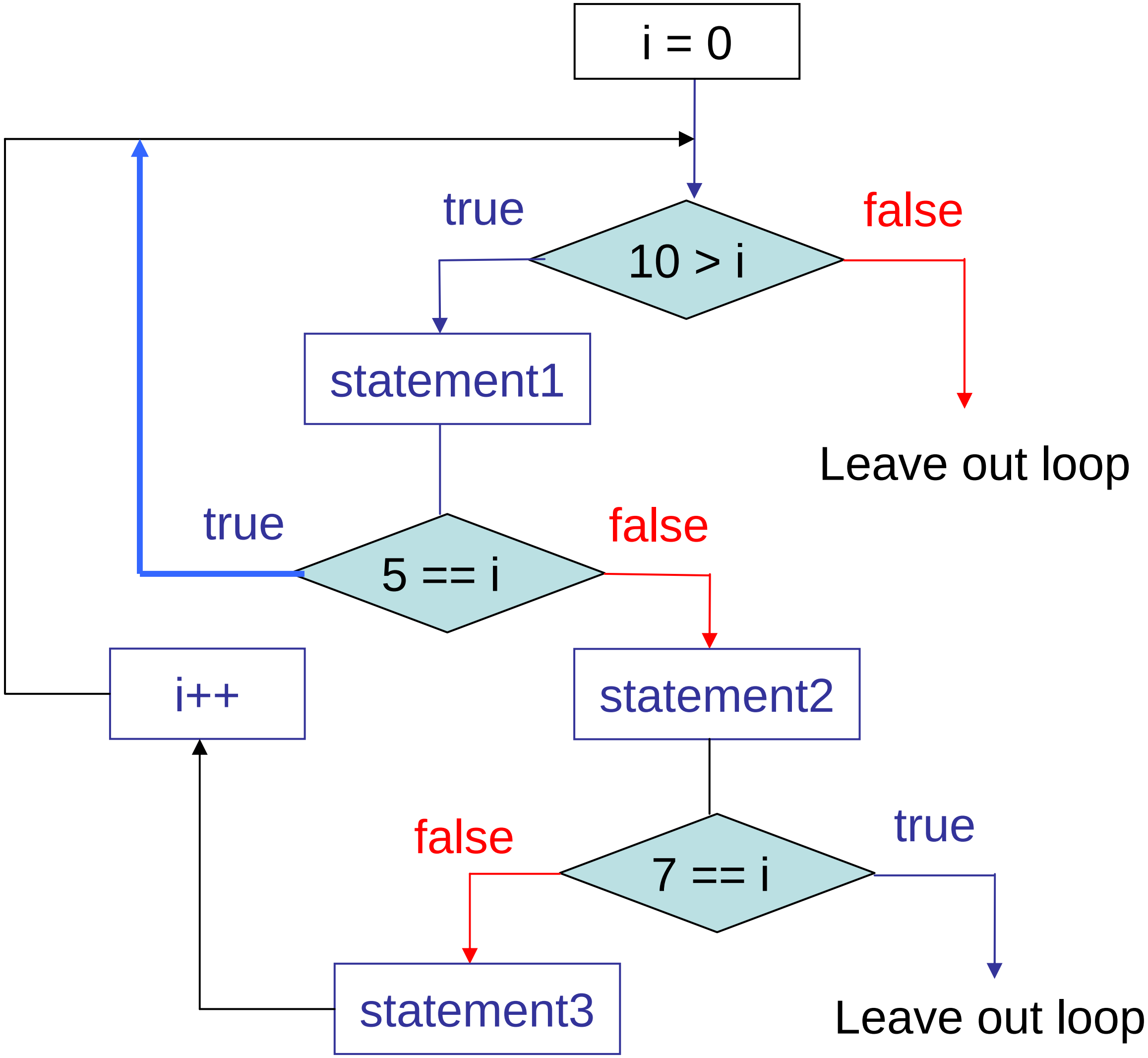
break and continue [1]

```
for ( i=0 ; 10 > i ; i++ ){
    statement1
    if ( 5 == i )  continue ;
    statement2
    if ( 7 == i )  break ;
    statement3
}
```

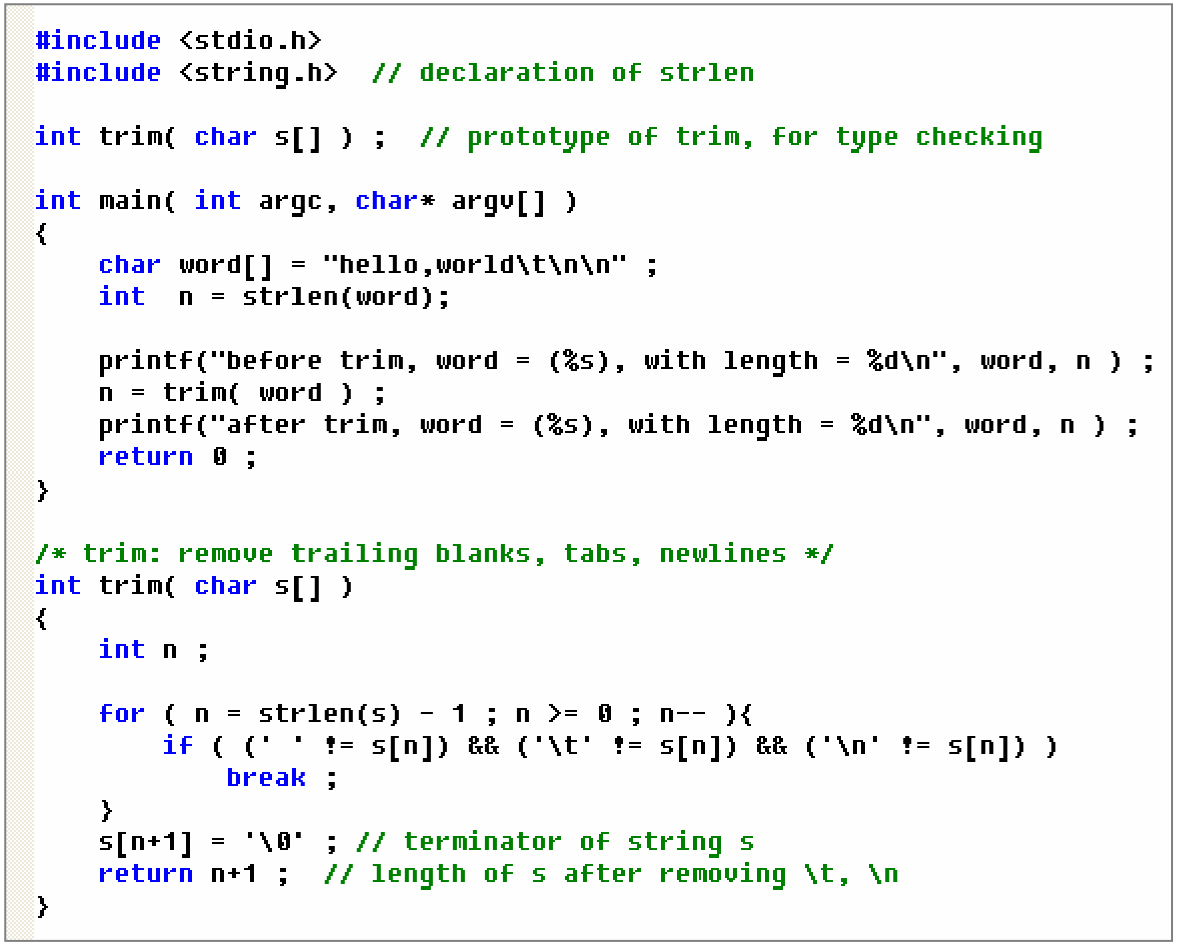


break and continue [2]

```
i = 0 ;
while ( 10 > i ){
      statement1
      if ( 5 == i )  continue ;
      statement2
      if ( 7 == i )  break ;
      statement3
      i++ ;
}
```



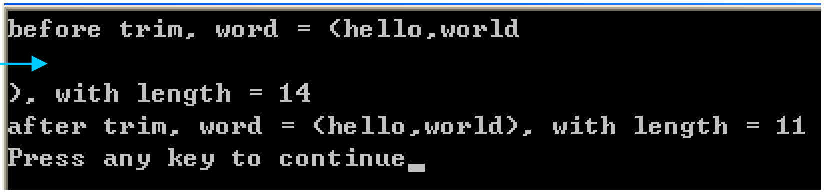
Example 5: trim in page 65



```c
#include <stdio.h>
#include <string.h>  // declaration of strlen

int trim( char s[] ) ;  // prototype of trim, for type checking

int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

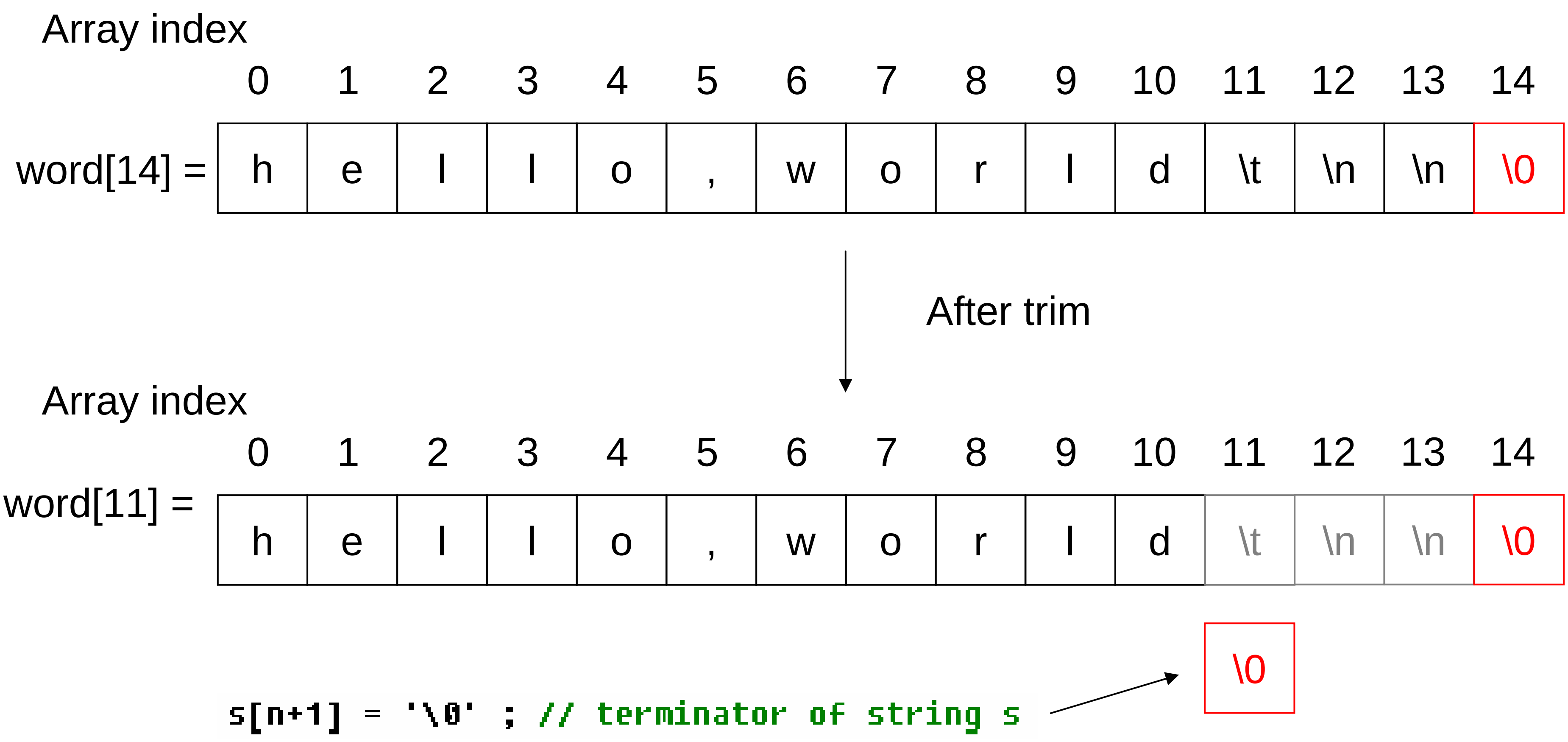
break is used to jump out the for-loop, not if-statement

Why empty line?



```
before trim, word = (hello,world

), with length = 14
after trim, word = (hello,world), with length = 11
Press any key to continue_
```

configuration before and after trim



Array index

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
word[14] =	h	e	l	l	o	,	w	o	r	l	d	\t	\n	\n	\0

After trim

Array index

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
word[11] =	h	e	l	l	o	,	w	o	r	l	d	\t	\n	\n	\0

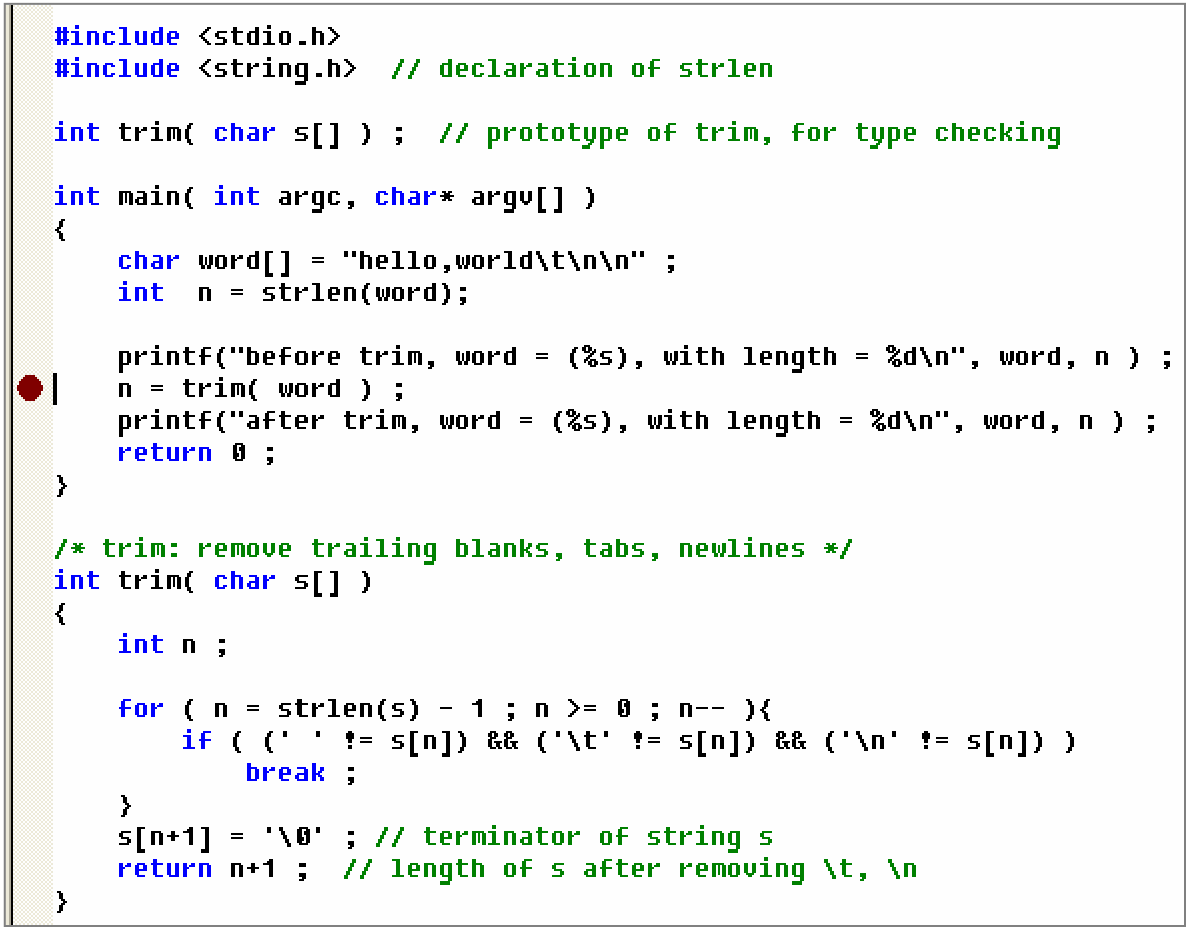
```
s[n+1] = '\0' ; // terminator of string s
```

OutLine

- expression and statement
- selection statement
- iteration statement
- Visual Studio debugger
- goto and labels

Use debugger to trace source codes [1]

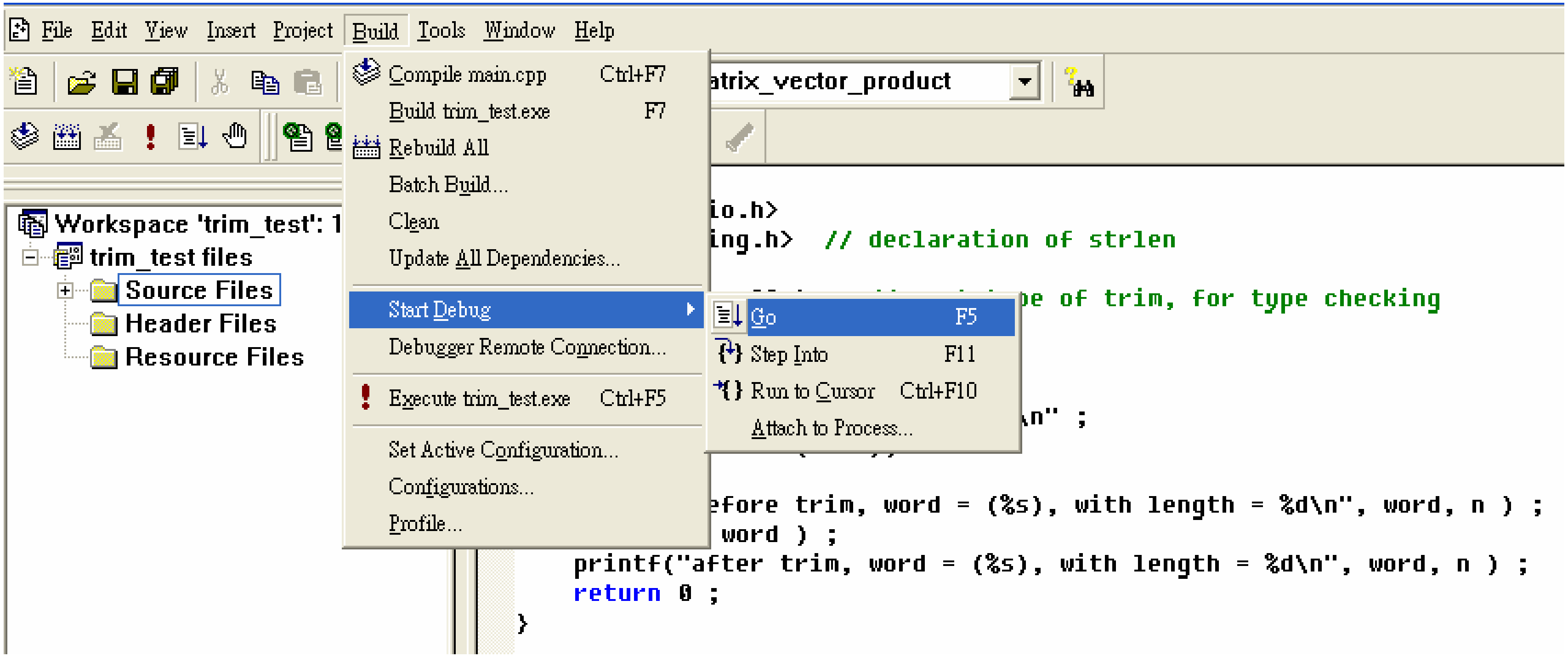
按 F9 產生中斷點 (breakpoint), 即紅色圓點



```
#include <stdio.h>
#include <string.h>  // declaration of strlen

int trim( char s[] ) ;  // prototype of trim, for type checking

int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

Use debugger to trace source codes [2]

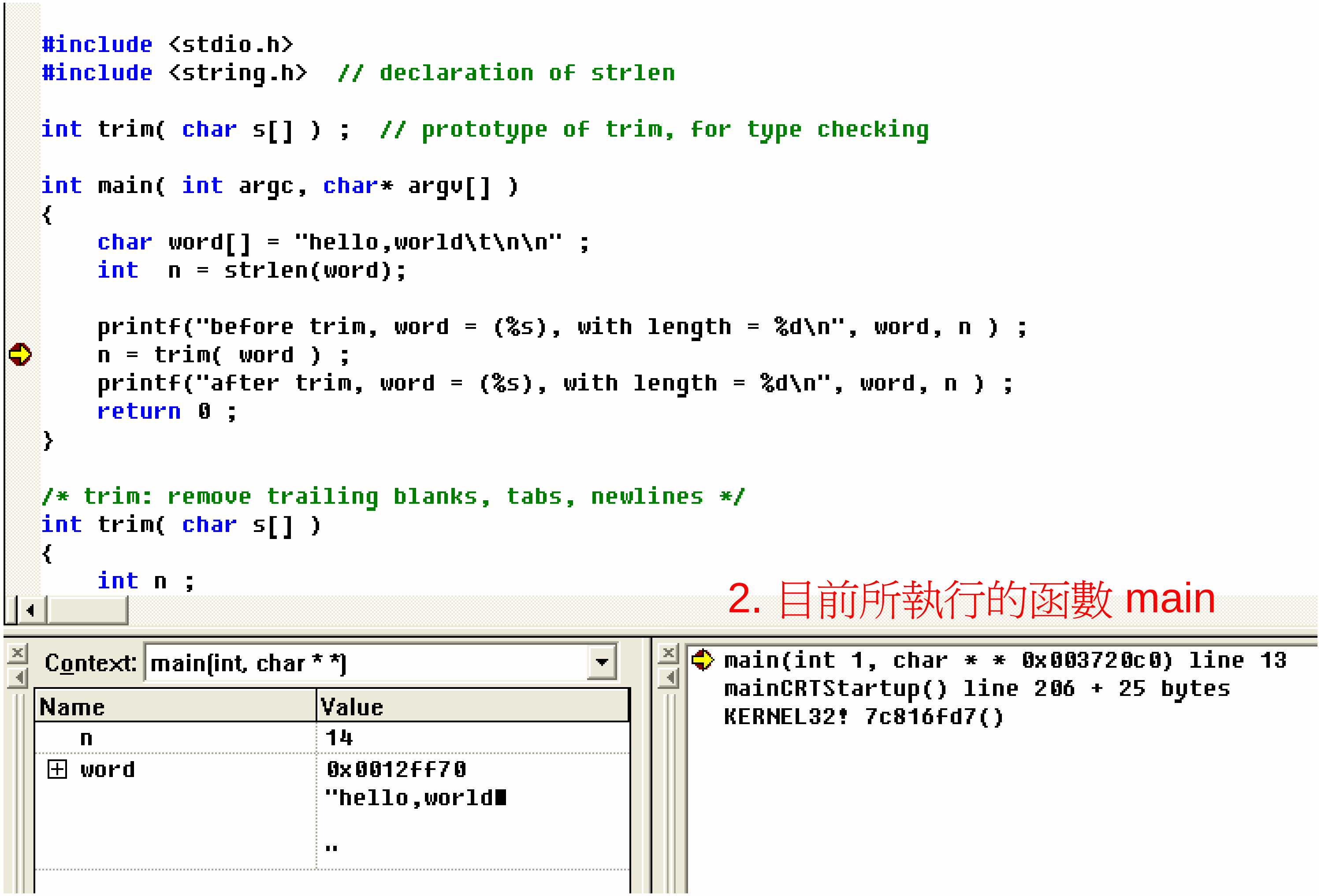
Build → Start Debug → Go



```
io.h>
ing.h>  // declaration of strlen

pe of trim, for type checking

\n" ;

efore trim, word = (%s), with length = %d\n", word, n ) ;
 word ) ;
   printf("after trim, word = (%s), with length = %d\n", word, n ) ;
   return 0 ;
}
```

Visual Studio debugger 會停在中斷點

Use debugger to trace source codes [3]



```c
#include <stdio.h>
#include <string.h>  // declaration of strlen

int trim( char s[] ) ;  // prototype of trim, for type checking

int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;
```

1. 停在此處中斷點

2. 目前所執行的函數 main

Context: main(int, char * *)

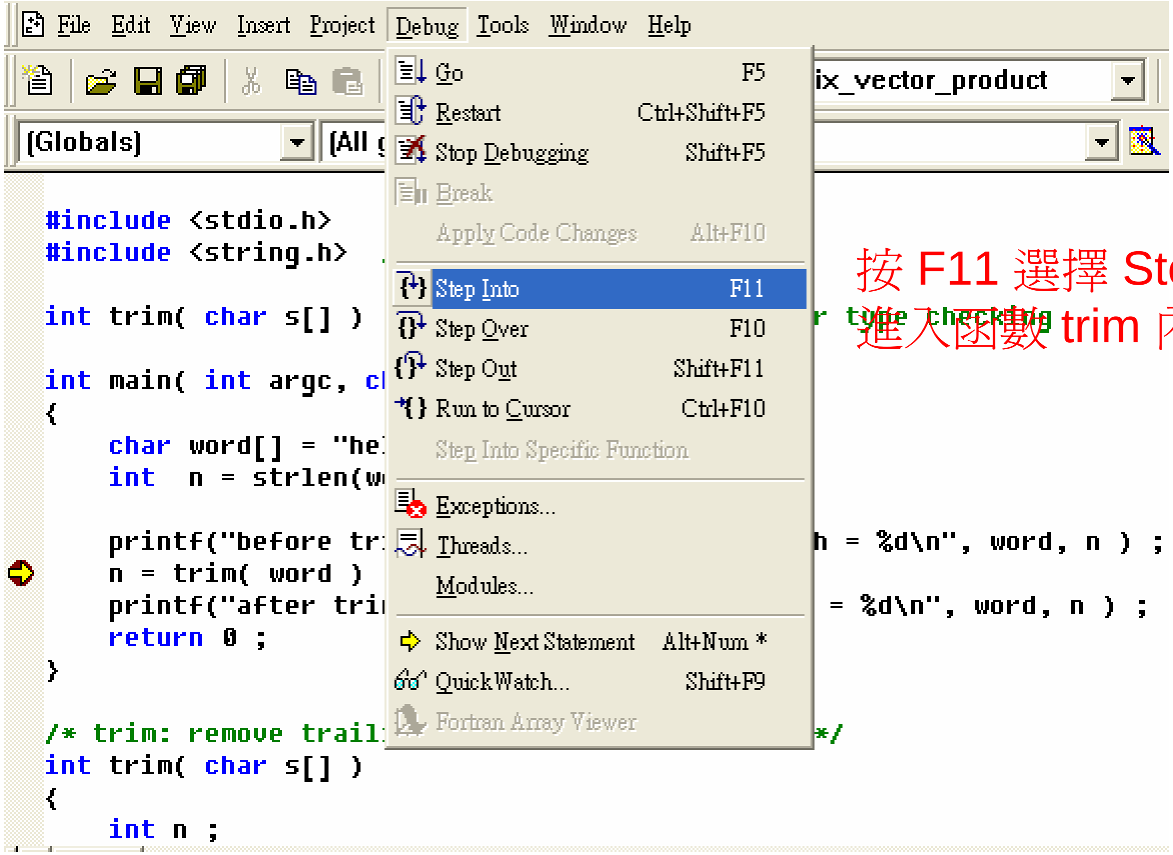
Name	Value
n	14
⊞ word	0x0012ff70 "hello,world▮ "

```
main(int 1, char * * 0x003720c0) line 13
mainCRTStartup() line 206 + 25 bytes
KERNEL32! 7c816fd7()
```

3. 函數main中的變數 n 和 word

Use debugger to trace source codes [4]

Debug → Step Into 或 直接按快速鍵 F11

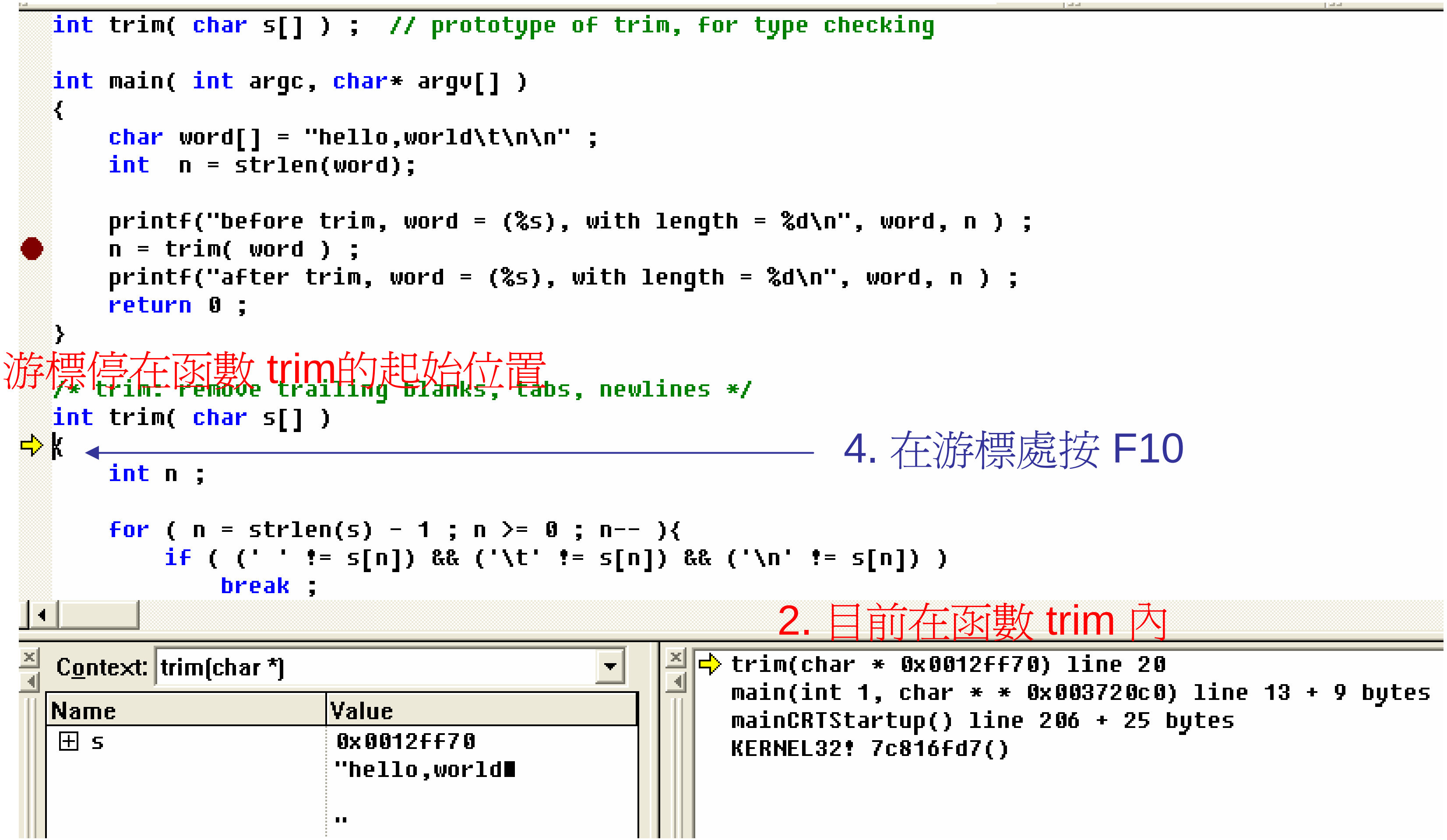


```
#include <stdio.h>
#include <string.h>  .

int trim( char s[] )             r type checking

int main( int argc, cl
{
    char word[] = "he
    int  n = strlen(w

    printf("before tr                    h = %d\n", word, n ) ;
    n = trim( word )
    printf("after tri                     = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trail                    */
int trim( char s[] )
{
    int n ;
```

按 F11 選擇 Step into, 即進入函數 trim 內

若按 F10 (step over) 則不進入函數內

Use debugger to trace source codes [5]



```
int trim( char s[] ) ;  // prototype of trim, for type checking

int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
```

Context: trim(char *)

Name	Value
⊞ s	0x0012ff70 "hello,world■ "

```
trim(char * 0x0012ff70) line 20
main(int 1, char * * 0x003720c0) line 13 + 9 bytes
mainCRTStartup() line 206 + 25 bytes
KERNEL32! 7c816fd7()
```

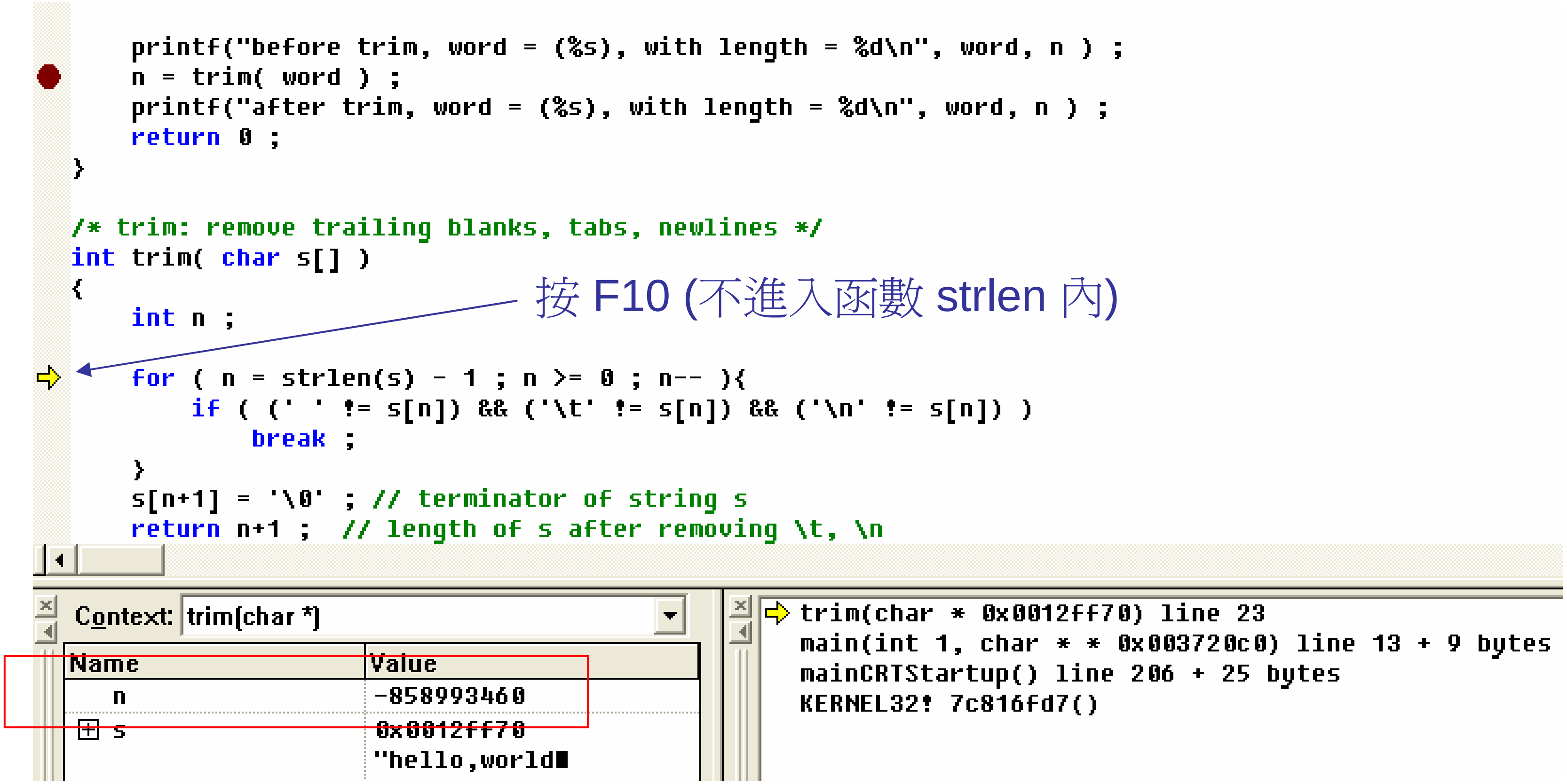
1. 游標停在函數 trim的起始位置

2. 目前在函數 trim 內

3. 輸入參數爲 character array s, 其值爲 “hello,world\t\n\n”

4. 在游標處按 F10

Use debugger to trace source codes [6]



```
    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
```

按 F10 (不進入函數 strlen 內)

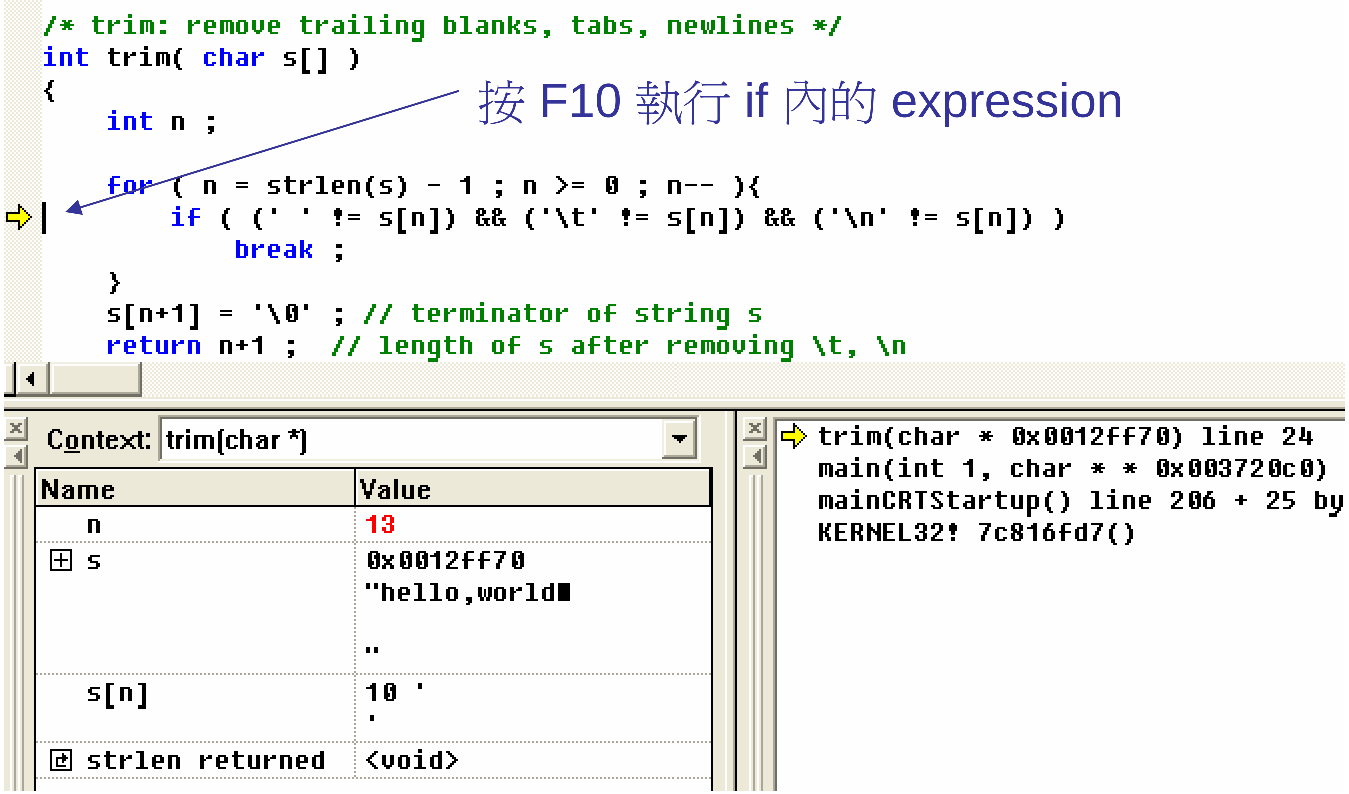
Context: trim(char *)

Name	Value
n	-858993460
⊞ s	0x0012ff70 "hello,world■

```
trim(char * 0x0012ff70) line 23
main(int 1, char * * 0x003720c0) line 13 + 9 bytes
mainCRTStartup() line 206 + 25 bytes
KERNEL32! 7c816fd7()
```

n 為 trim 的區域變數, 但只執行其宣告, 所以其值為亂碼

Use debugger to trace source codes [7]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
```

按 F10 執行 if 內的 expression

Context: trim(char *)

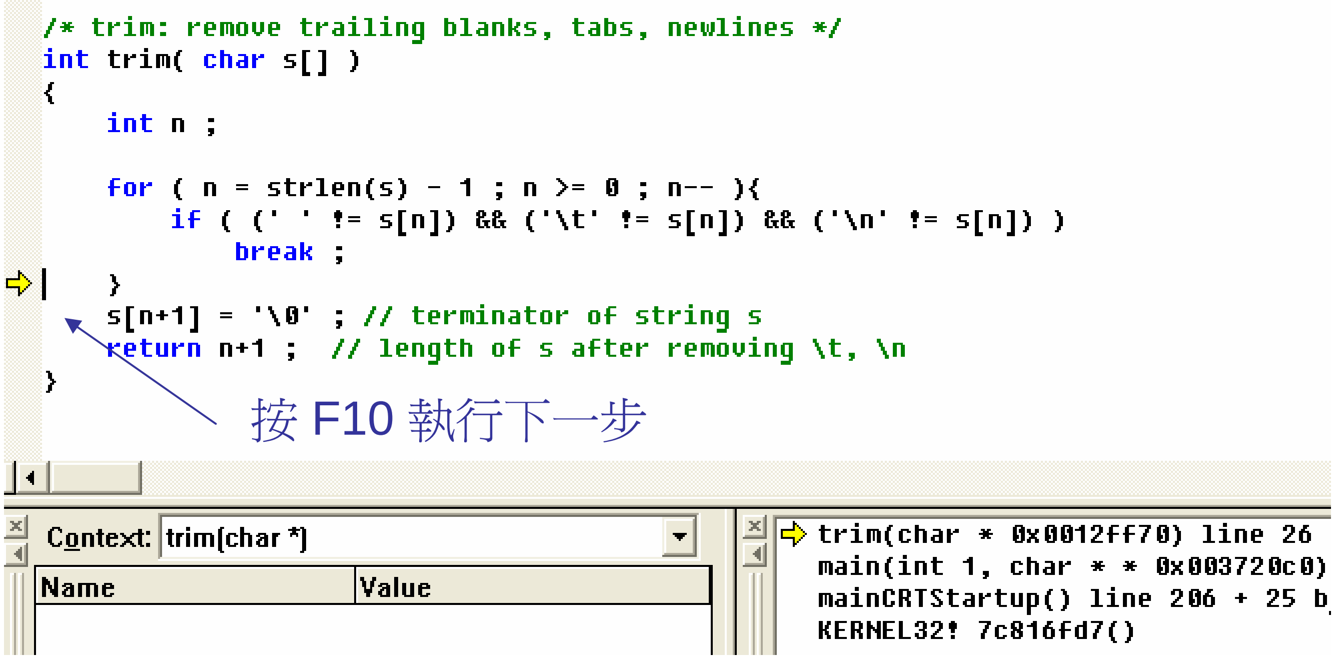
Name	Value
n	13
⊞ s	0x0012ff70 "hello,world■ .."
s[n]	10 ' '
strlen returned	<void>

```
trim(char * 0x0012ff70) line 24
main(int 1, char * * 0x003720c0)
mainCRTStartup() line 206 + 25 by
KERNEL32! 7c816fd7()
```

1. 執行 n = strlen(s) – 1, 所以 n = 13

2. s[n] = s[13] = ‘\n’ = 10 (decimal number)

Use debugger to trace source codes [8]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

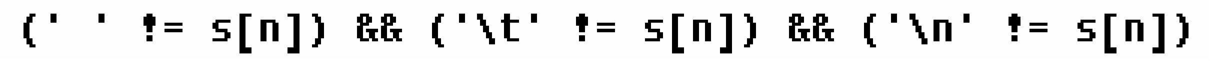
按 F10 執行下一步

Context: trim(char *)

Name	Value

```
trim(char * 0x0012ff70) line 26
main(int 1, char * * 0x003720c0)
mainCRTStartup() line 206 + 25 b
KERNEL32! 7c816fd7()
```

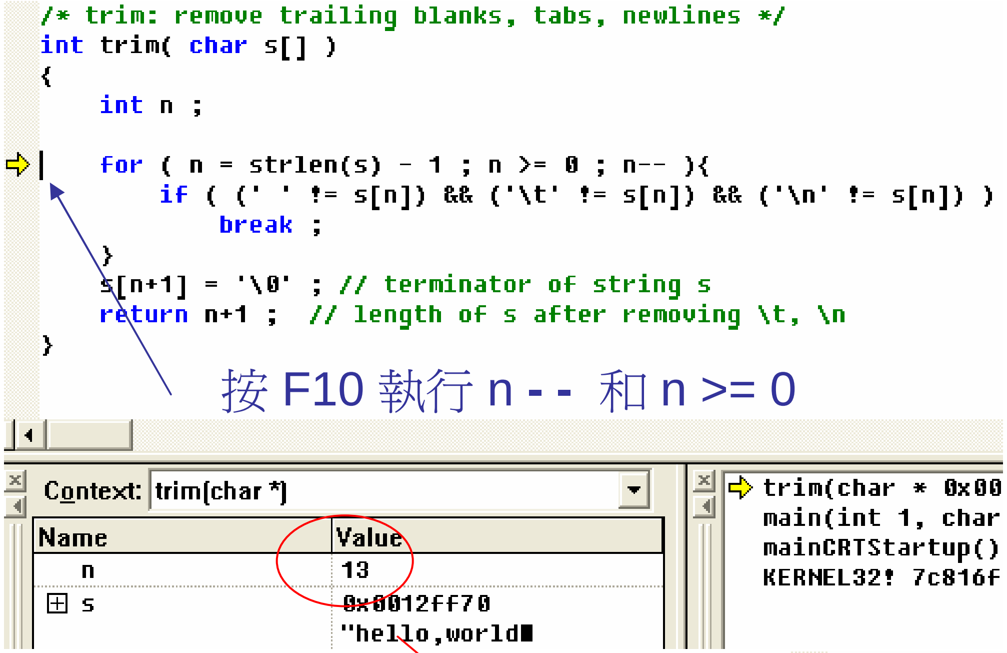
因爲 s[n] = s[13] = ‘\n’ 不滿足條件式



```
(' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n])
```

所以不執行 break statement

Use debugger to trace source codes [9]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

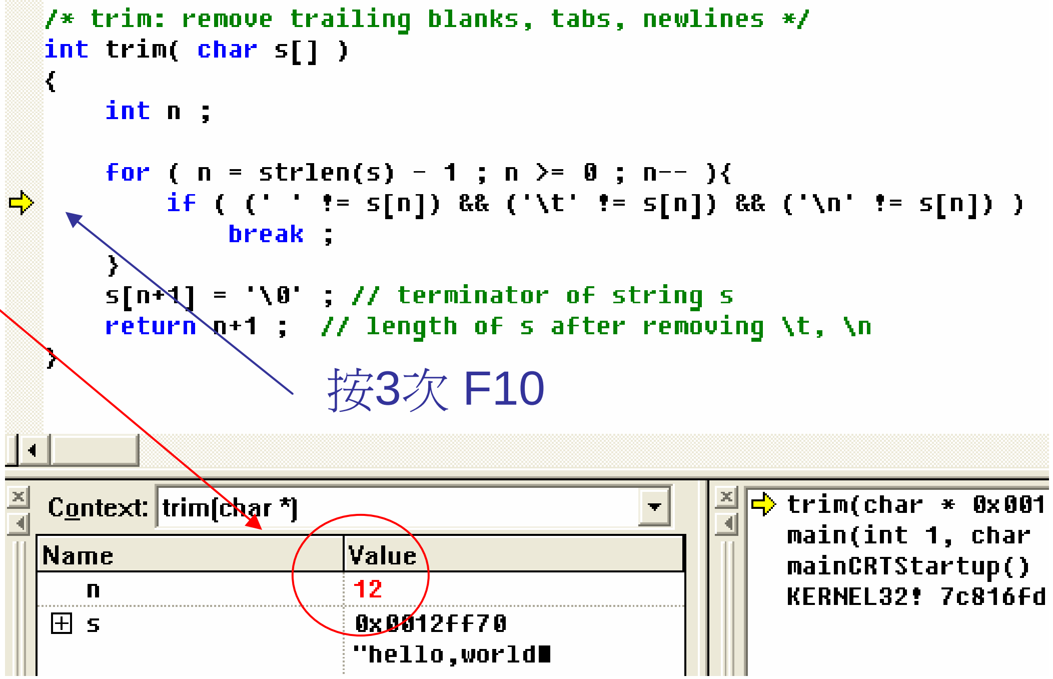
按 F10 執行 n - - 和 n >= 0

Context: trim(char *)

Name	Value
n	13
⊞ s	0x0012ff70 "hello,world■

```
trim(char * 0x00
main(int 1, char
mainCRTStartup()
KERNEL32! 7c816f
```

執行 n - - , 所以 n = 12



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

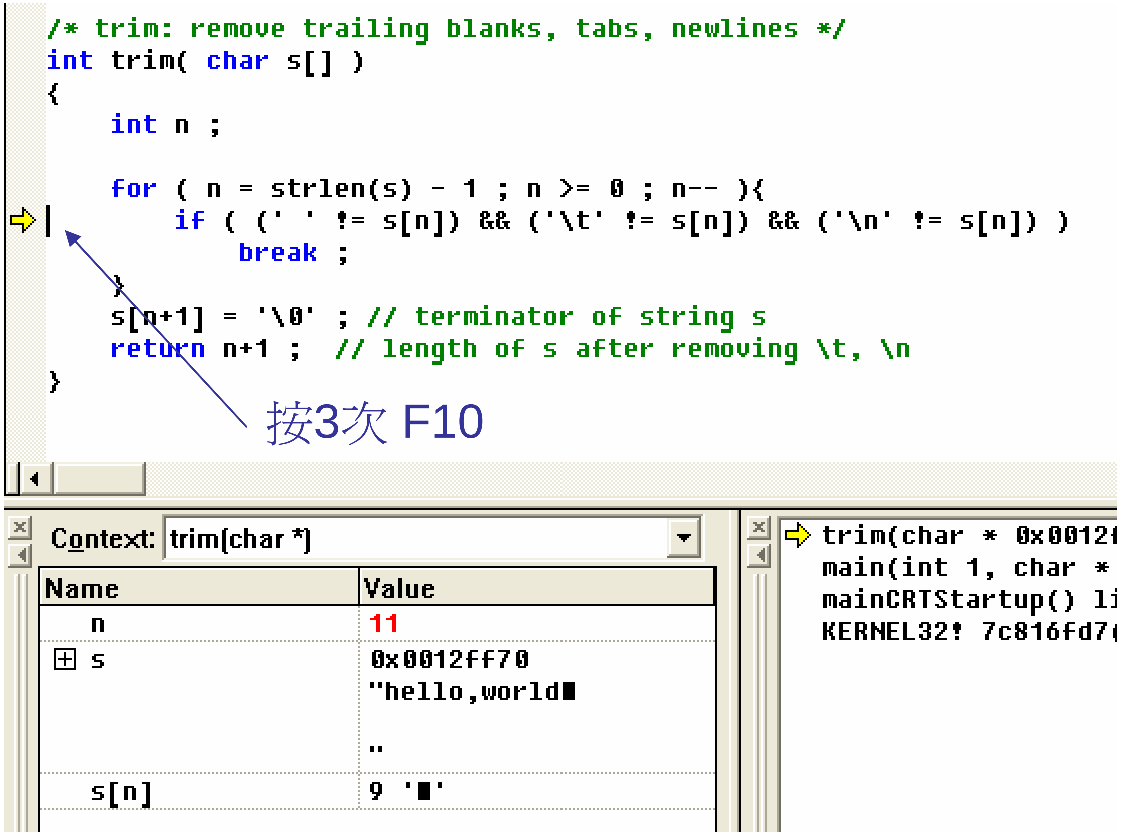
按3次 F10

Context: trim(char *)

Name	Value
n	12
⊞ s	0x0012ff70 "hello,world■

```
trim(char * 0x001
main(int 1, char
mainCRTStartup()
KERNEL32! 7c816fd
```

Use debugger to trace source codes [10]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

按3次 F10

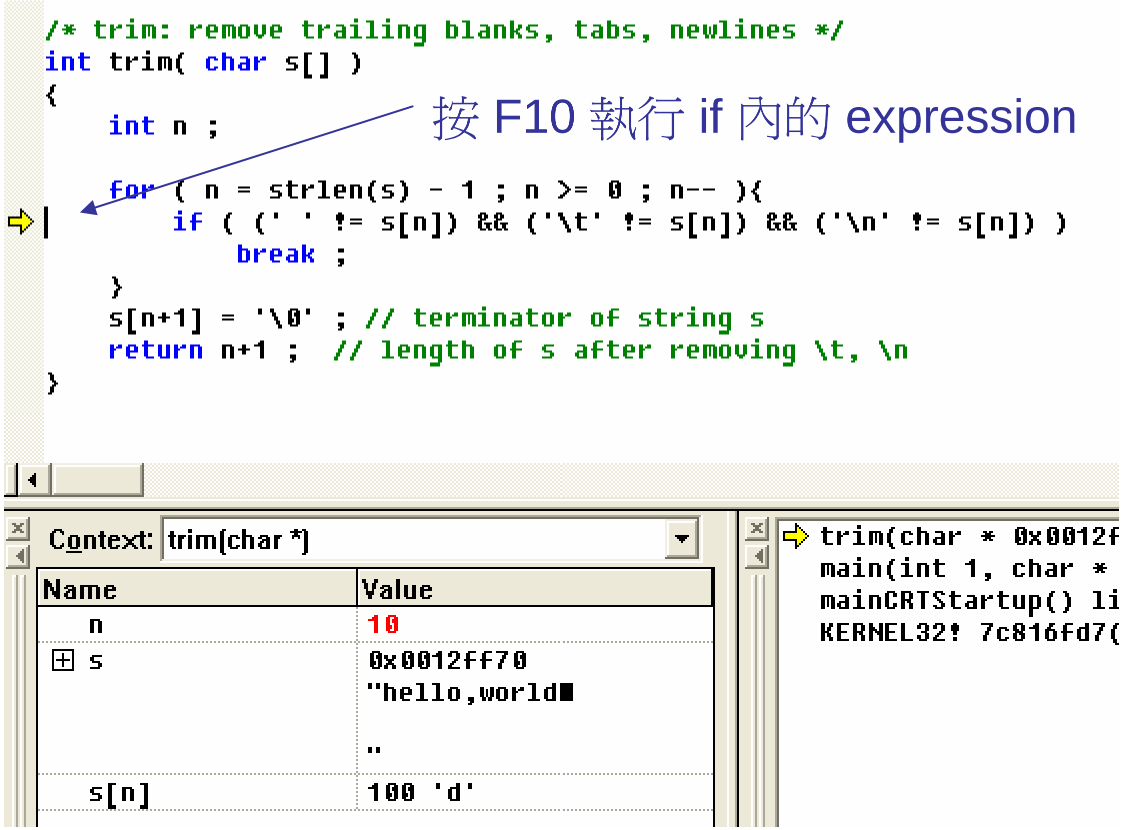
Context: trim(char *)

Name	Value
n	11
⊞ s	0x0012ff70 "hello,world▮ "
s[n]	9 '▮'

```
trim(char * 0x0012f
main(int 1, char *
mainCRTStartup() li
KERNEL32! 7c816fd7(
```

n =11 且 s[n] = s[11] = ‘\t’ = 9 (decimal)

Use debugger to trace source codes [11]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

按 F10 執行 if 內的 expression

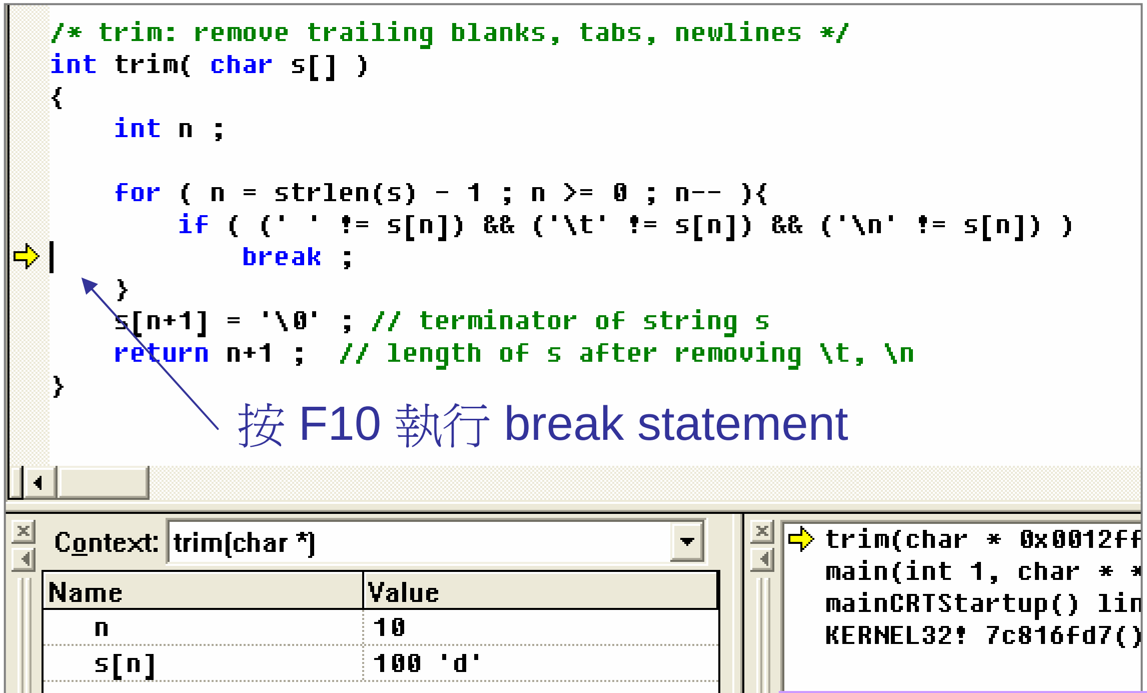
Name	Value
n	10
⊞ s	0x0012ff70 "hello,world█ ..
s[n]	100 'd'

```
trim(char * 0x0012f
main(int 1, char *
mainCRTStartup() li
KERNEL32! 7c816fd7(
```

n =10 且 s[n] = s[10] = ‘d’ = 100 (decimal)

s[10] = ‘d’ 滿足 if 內的條件式

Use debugger to trace source codes [12]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

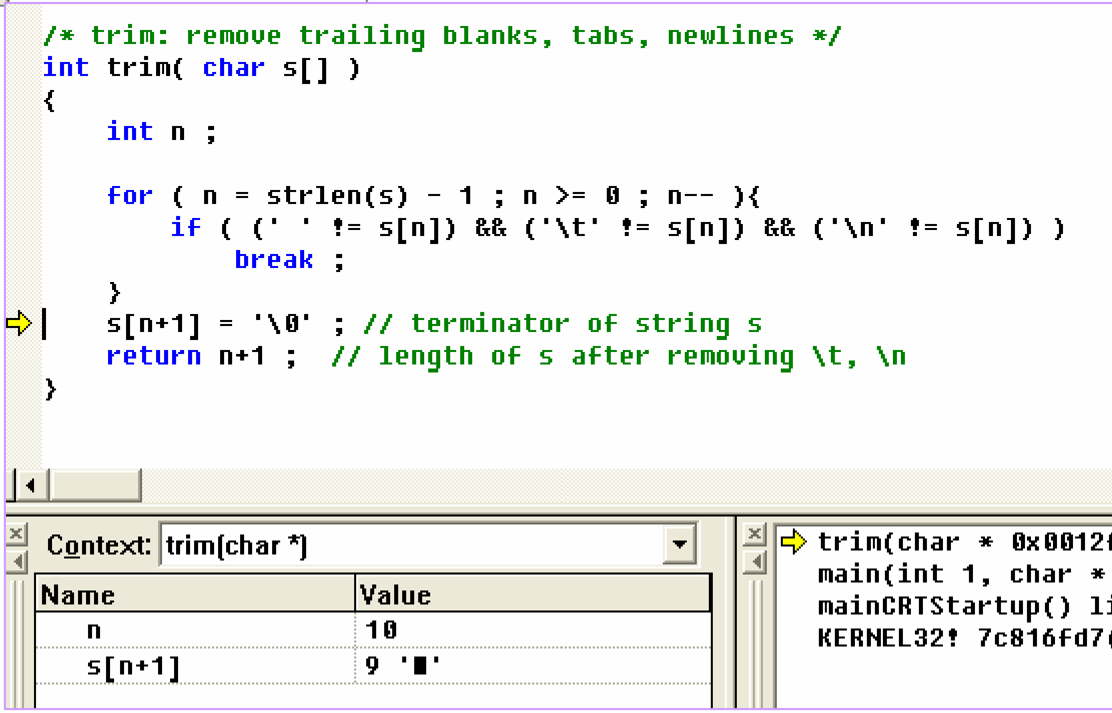
按 F10 執行 break statement

Context: trim(char *)

Name	Value
n	10
s[n]	100 'd'

```
trim(char * 0x0012ff
main(int 1, char * *
mainCRTStartup() lin
KERNEL32! 7c816fd7()
```

按 F10 覆蓋 s[11] = ‘\t’



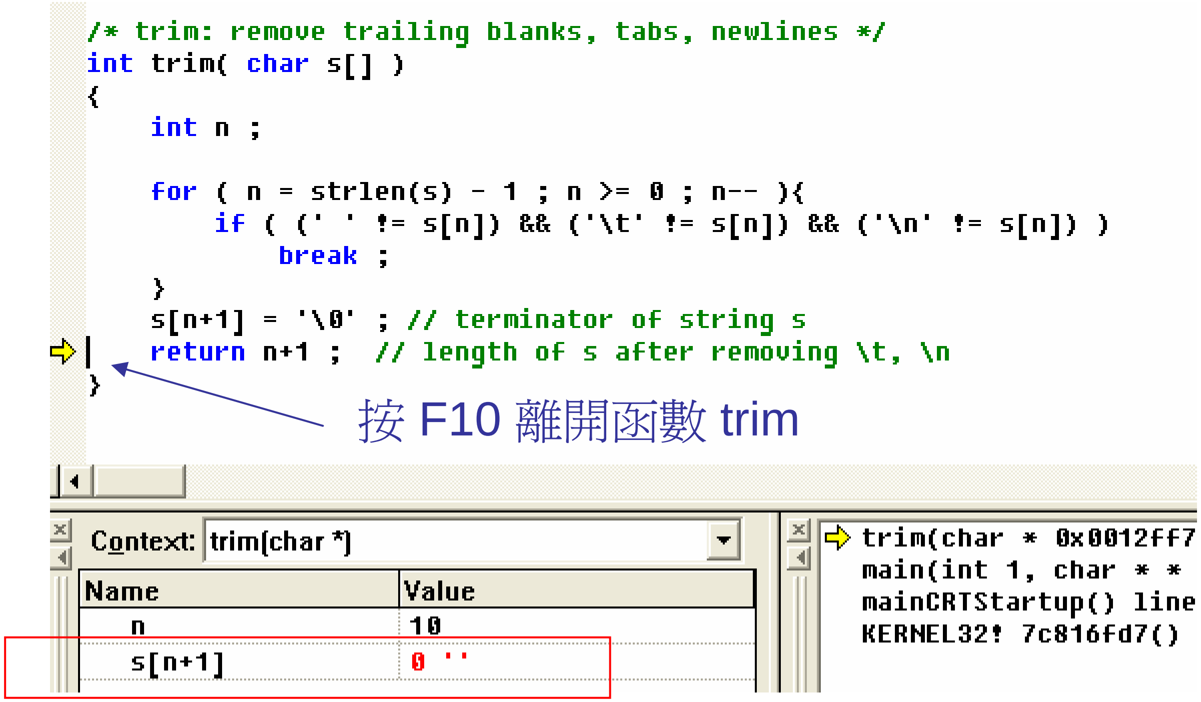
```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

Context: trim(char *)

Name	Value
n	10
s[n+1]	9 '■'

```
trim(char * 0x0012f
main(int 1, char *
mainCRTStartup() li
KERNEL32! 7c816fd7(
```

Use debugger to trace source codes [13]



```
/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

按 F10 離開函數 trim

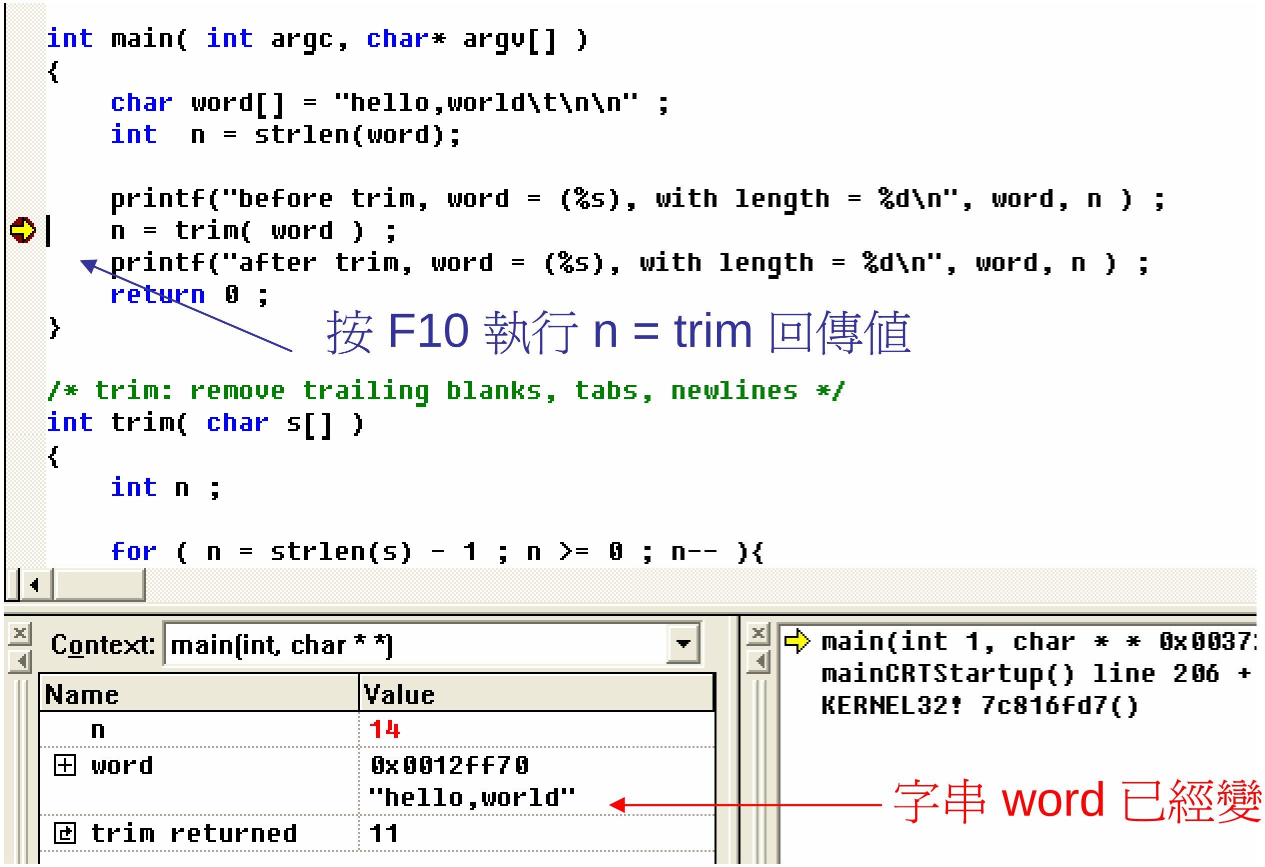
Context: trim(char *)

Name	Value
n	10
s[n+1]	0 ''

```
trim(char * 0x0012ff7
main(int 1, char * *
mainCRTStartup() line
KERNEL32! 7c816fd7()
```

s[11] = ‘\0’ = 0 所以字串 s = “hello,world”

Use debugger to trace source codes [14]



```
int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
```

按 F10 執行 n = trim 回傳值

Context: main(int, char * *)

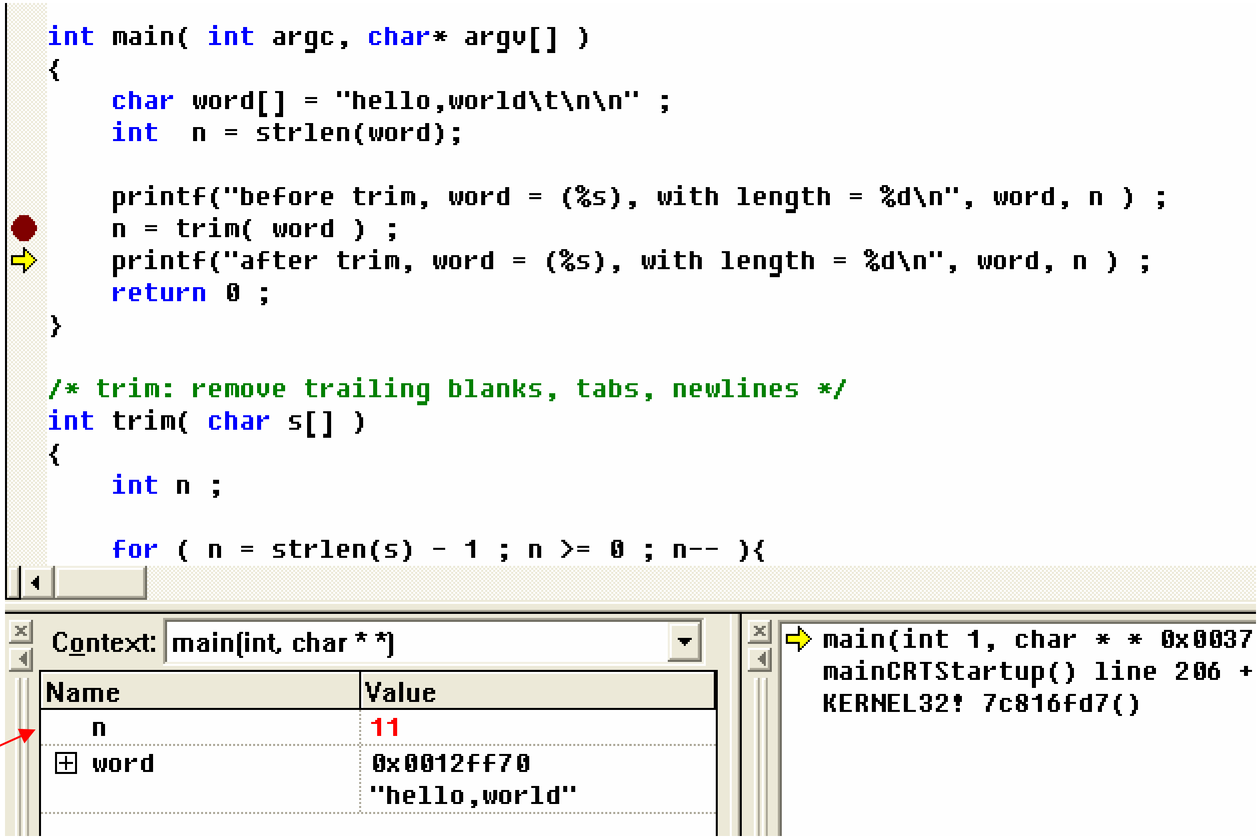
Name	Value
n	14
⊞ word	0x0012ff70 "hello,world"
trim returned	11

```
main(int 1, char * * 0x0037
mainCRTStartup() line 206 +
KERNEL32! 7c816fd7()
```

字串 word 已經變成 “hello,world”

函數 trim 回傳 n+1 = 10+1 = 11, 但函數 main中的變數 n 依舊是 14

Use debugger to trace source codes [15]



```
int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
```

Context: main(int, char * *)

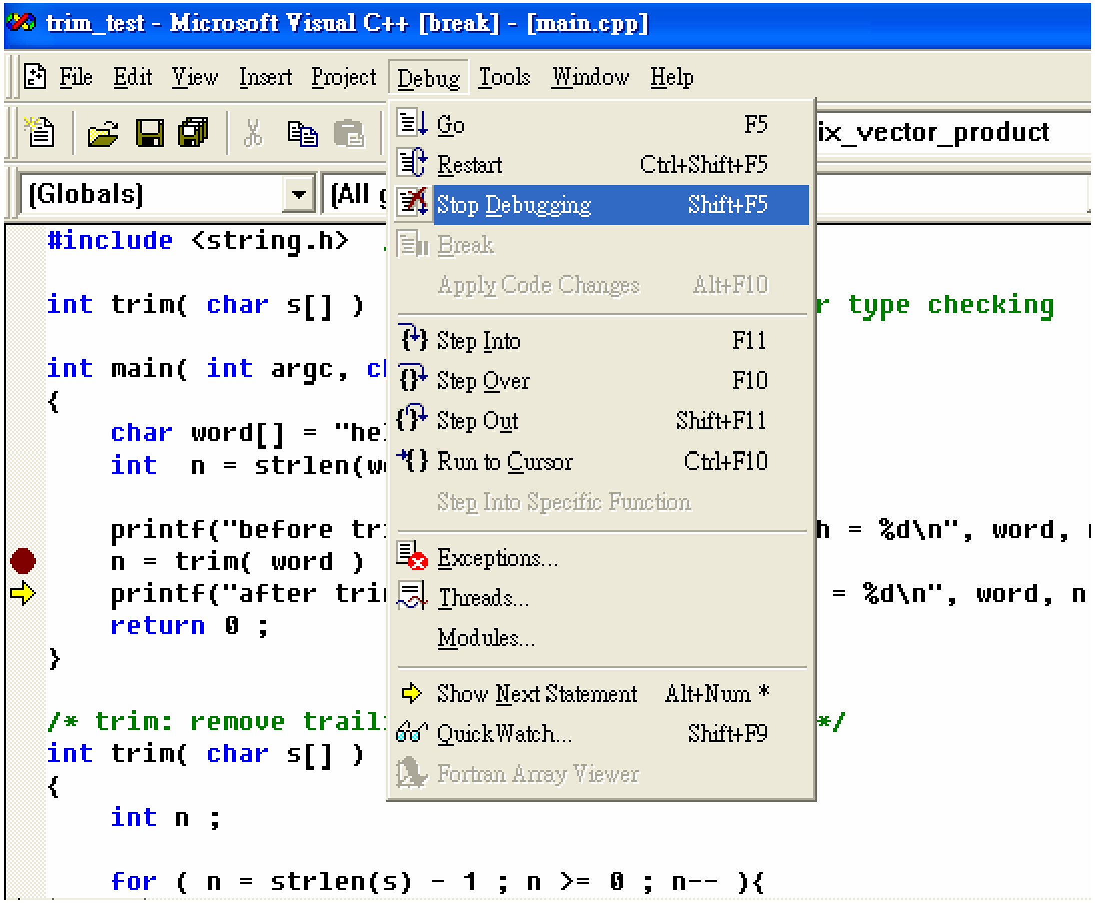
Name	Value
n	11
⊞ word	0x0012ff70 "hello,world"

```
main(int 1, char * * 0x0037
mainCRTStartup() line 206 +
KERNEL32! 7c816fd7()
```

n = 11 (即 trim 的回傳值)

Use debugger to trace source codes [16]

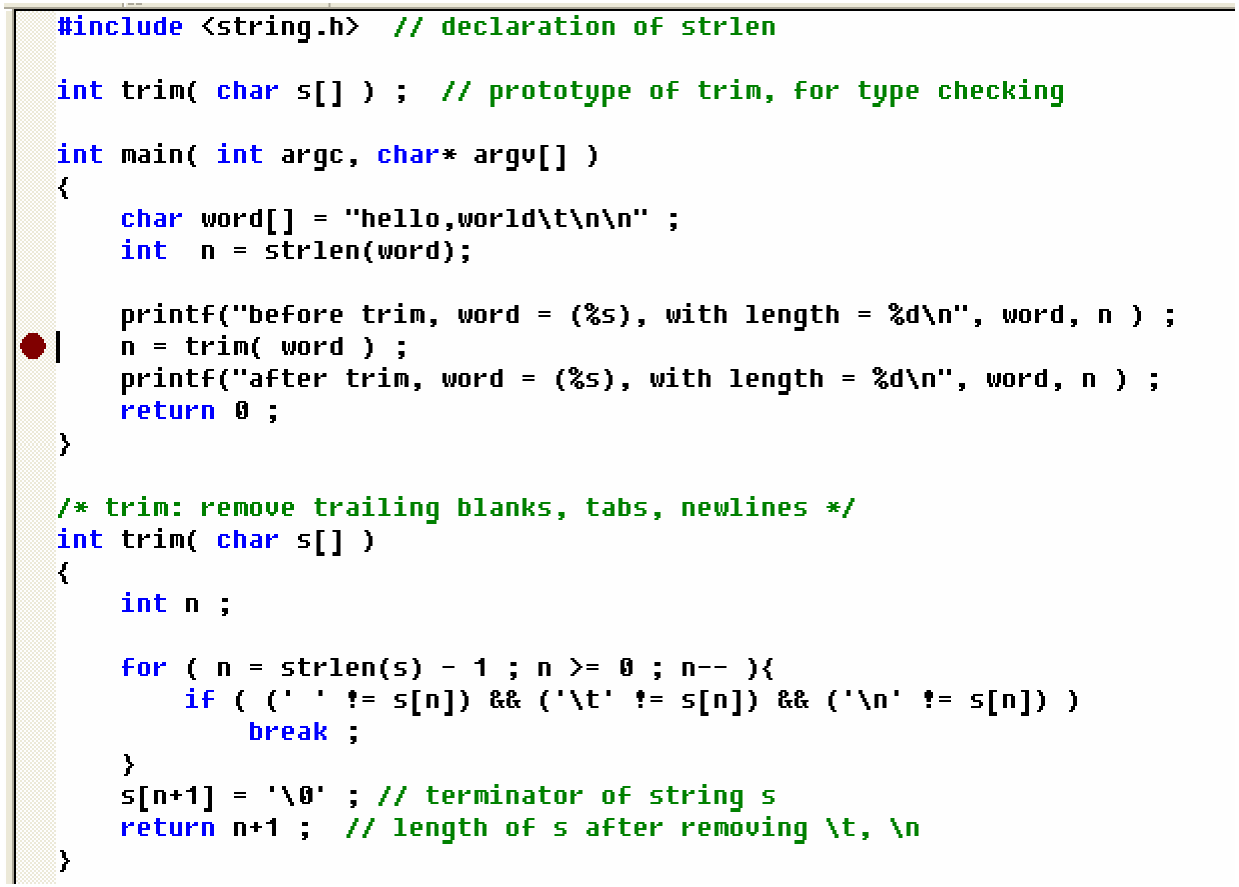
Debug → Stop Debugging



```
#include <string.h>   .

int trim( char s[] )                                    r type checking

int main( int argc, cl
{
    char word[] = "he
    int  n = strlen(w

    printf("before tr                                   h = %d\n", word, 
    n = trim( word )
    printf("after tri                                    = %d\n", word, n
    return 0 ;
}

/* trim: remove trail                                   */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
```

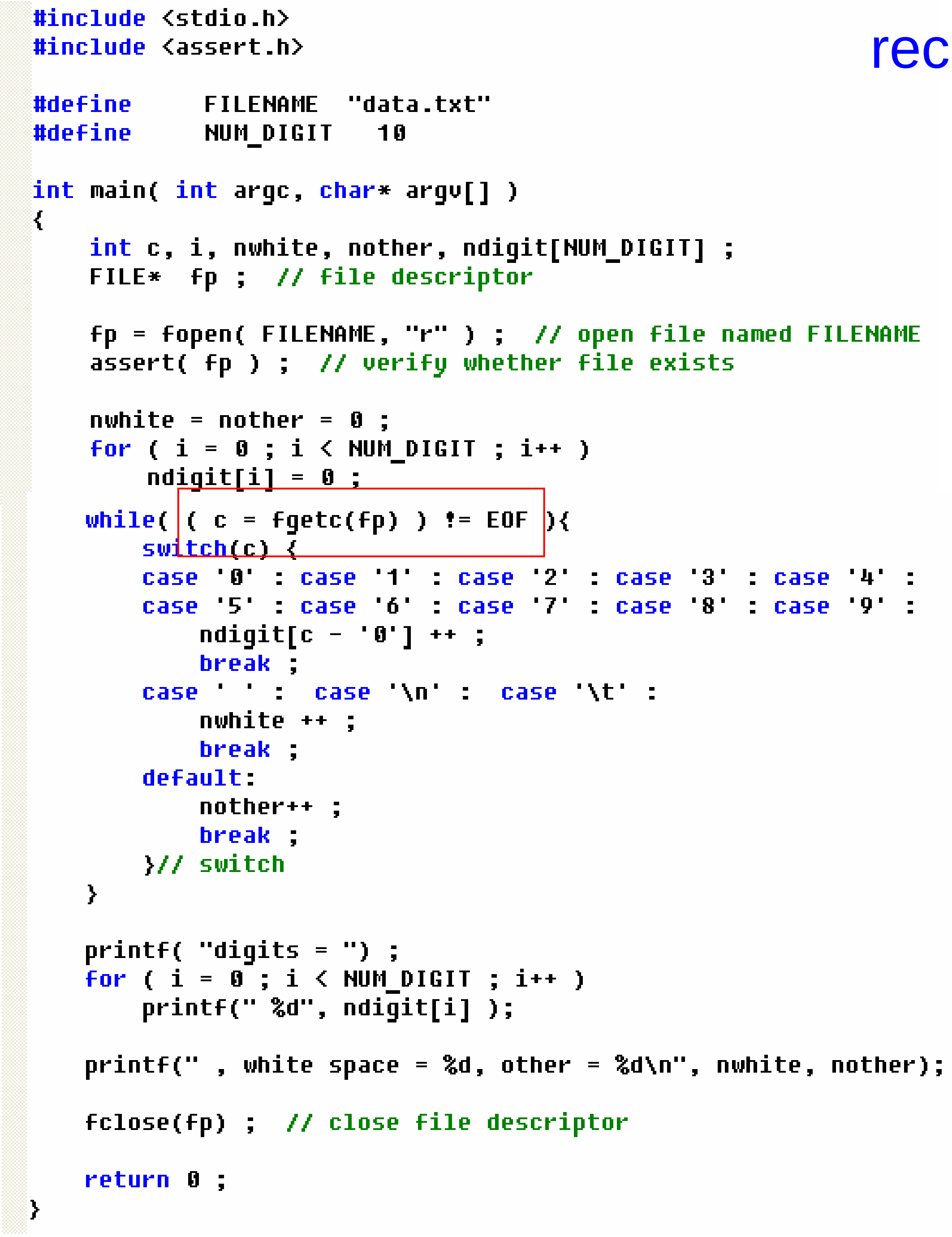
Use debugger to trace source codes [17]

按 F9 取消中斷點



```
#include <string.h>  // declaration of strlen

int trim( char s[] ) ;  // prototype of trim, for type checking

int main( int argc, char* argv[] )
{
    char word[] = "hello,world\t\n\n" ;
    int  n = strlen(word);

    printf("before trim, word = (%s), with length = %d\n", word, n ) ;
    n = trim( word ) ;
    printf("after trim, word = (%s), with length = %d\n", word, n ) ;
    return 0 ;
}

/* trim: remove trailing blanks, tabs, newlines */
int trim( char s[] )
{
    int n ;

    for ( n = strlen(s) - 1 ; n >= 0 ; n-- ){
        if ( (' ' != s[n]) && ('\t' != s[n]) && ('\n' != s[n]) )
            break ;
    }
    s[n+1] = '\0' ; // terminator of string s
    return n+1 ;  // length of s after removing \t, \n
}
```

recall example 4



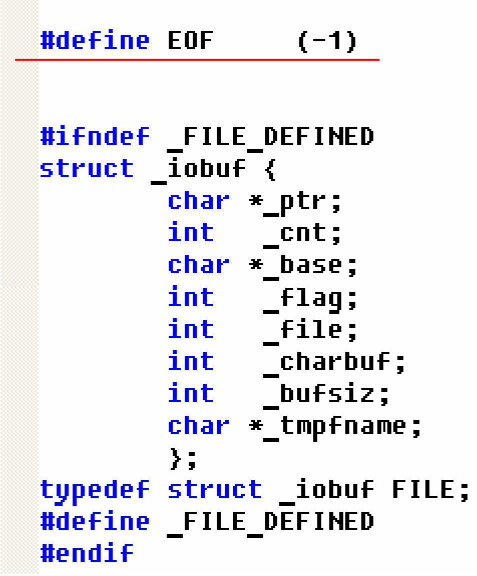
```
#include <stdio.h>
#include <assert.h>

#define     FILENAME  "data.txt"
#define     NUM_DIGIT   10

int main( int argc, char* argv[] )
{
    int c, i, nwhite, nother, ndigit[NUM_DIGIT] ;
    FILE*  fp ;  // file descriptor

    fp = fopen( FILENAME, "r" ) ;  // open file named FILENAME
    assert( fp ) ;  // verify whether file exists

    nwhite = nother = 0 ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        ndigit[i] = 0 ;

    while( ( c = fgetc(fp) ) != EOF ){
        switch(c) {
        case '0' : case '1' : case '2' : case '3' : case '4' :
        case '5' : case '6' : case '7' : case '8' : case '9' :
            ndigit[c - '0'] ++ ;
            break ;
        case ' ' :  case '\n' :  case '\t' :
            nwhite ++ ;
            break ;
        default:
            nother++ ;
            break ;
        }// switch
    }

    printf( "digits = ") ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        printf(" %d", ndigit[i] );

    printf(" , white space = %d, other = %d\n", nwhite, nother);

    fclose(fp) ;  // close file descriptor

    return 0 ;
}
```

int fgetc(FILE *stream)

fgetc returns the next character of stream as an unsigned char (converted to an int), or EOF if end of file or error occurs.

see page 246

stdio.h



```
#define EOF     (-1)


#ifndef _FILE_DEFINED
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
#define _FILE_DEFINED
#endif
```

(c = fgetc(fp)) != EOF

Step 1: c = fgetc(fp)

Step 2: c != EOF

c : l-value

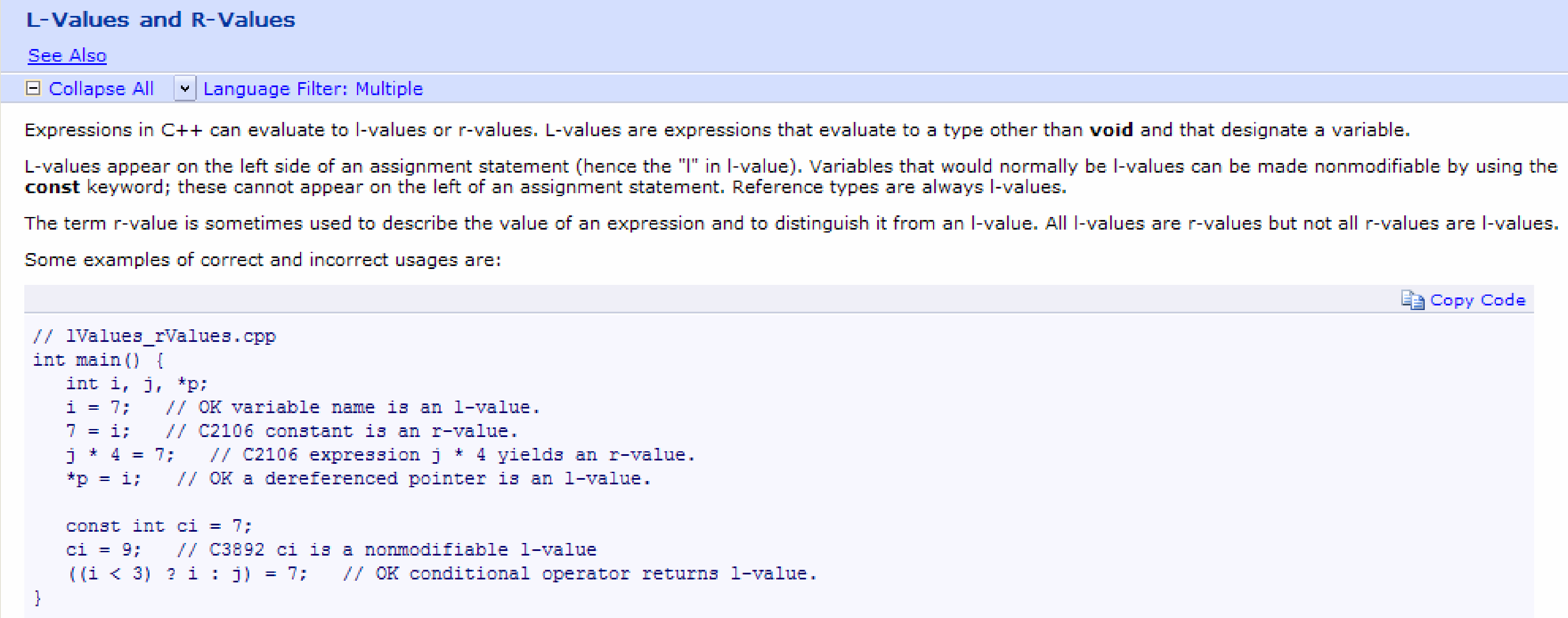
expression “c = fgetc(fp)” has r-value c

An object is a named region of storage

An l-value is an expression referring to an **object**

R-value : value

L-value : location



L-Values and R-Values

Expressions in C++ can evaluate to l-values or r-values. L-values are expressions that evaluate to a type other than **void** and that designate a variable.

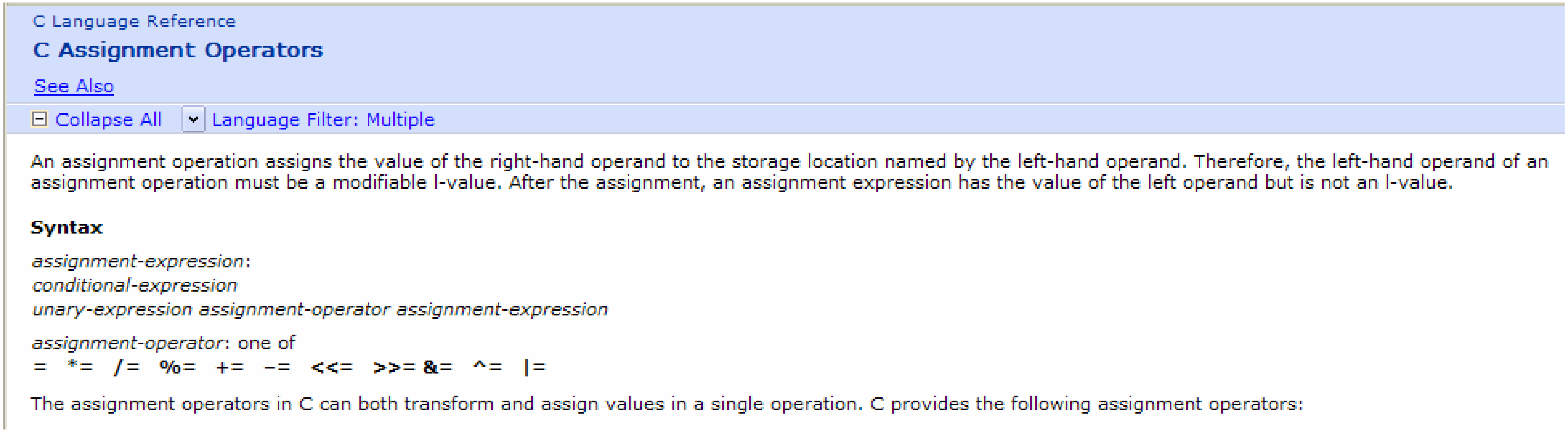
L-values appear on the left side of an assignment statement (hence the "l" in l-value). Variables that would normally be l-values can be made nonmodifiable by using the **const** keyword; these cannot appear on the left of an assignment statement. Reference types are always l-values.

The term r-value is sometimes used to describe the value of an expression and to distinguish it from an l-value. All l-values are r-values but not all r-values are l-values.

Some examples of correct and incorrect usages are:

```
// lValues_rValues.cpp
int main() {
   int i, j, *p;
   i = 7;   // OK variable name is an l-value.
   7 = i;   // C2106 constant is an r-value.
   j * 4 = 7;   // C2106 expression j * 4 yields an r-value.
   *p = i;   // OK a dereferenced pointer is an l-value.

   const int ci = 7;
   ci = 9;   // C3892 ci is a nonmodifiable l-value
   ((i < 3) ? i : j) = 7;   // OK conditional operator returns l-value.
}
```

inheritance of L-value in assignment statement



C Language Reference

C Assignment Operators

See Also

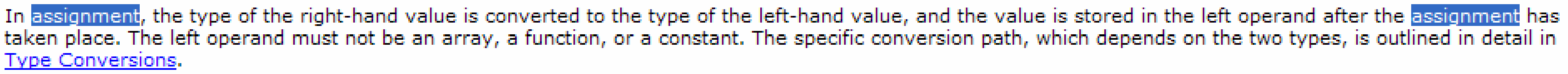
An assignment operation assigns the value of the right-hand operand to the storage location named by the left-hand operand. Therefore, the left-hand operand of an assignment operation must be a modifiable l-value. After the assignment, an assignment expression has the value of the left operand but is not an l-value.

Syntax

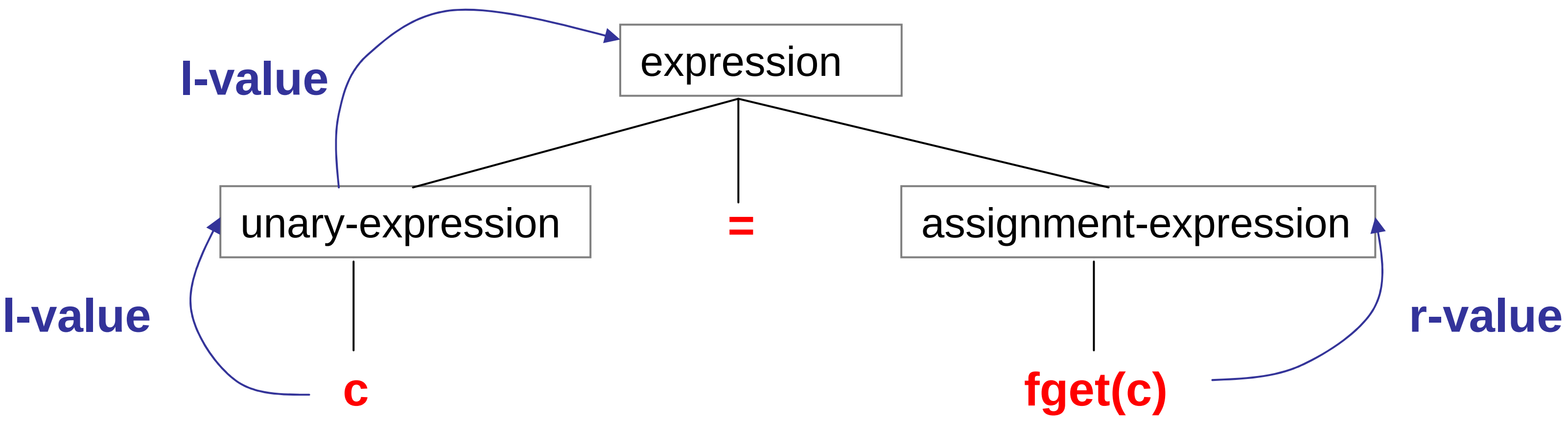
assignment-expression:
conditional-expression
unary-expression assignment-operator assignment-expression

assignment-operator: one of
= *= /= %= += –= <<= >>= &= ^= |=

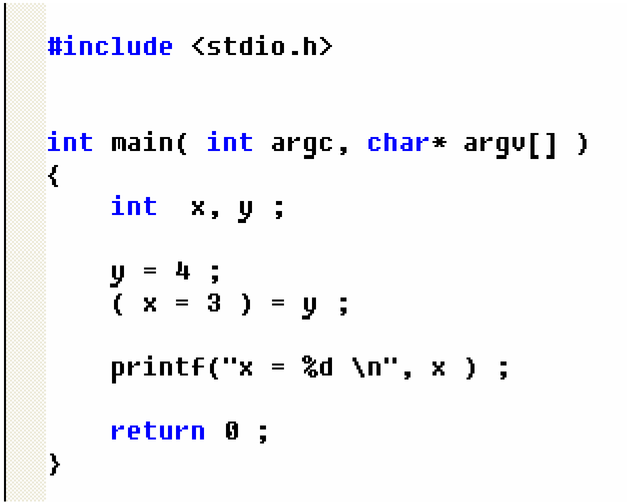
The assignment operators in C can both transform and assign values in a single operation. C provides the following assignment operators:



In assignment, the type of the right-hand value is converted to the type of the left-hand value, and the value is stored in the left operand after the assignment has taken place. The left operand must not be an array, a function, or a constant. The specific conversion path, which depends on the two types, is outlined in detail in Type Conversions.



Example 6: L-value test [1]

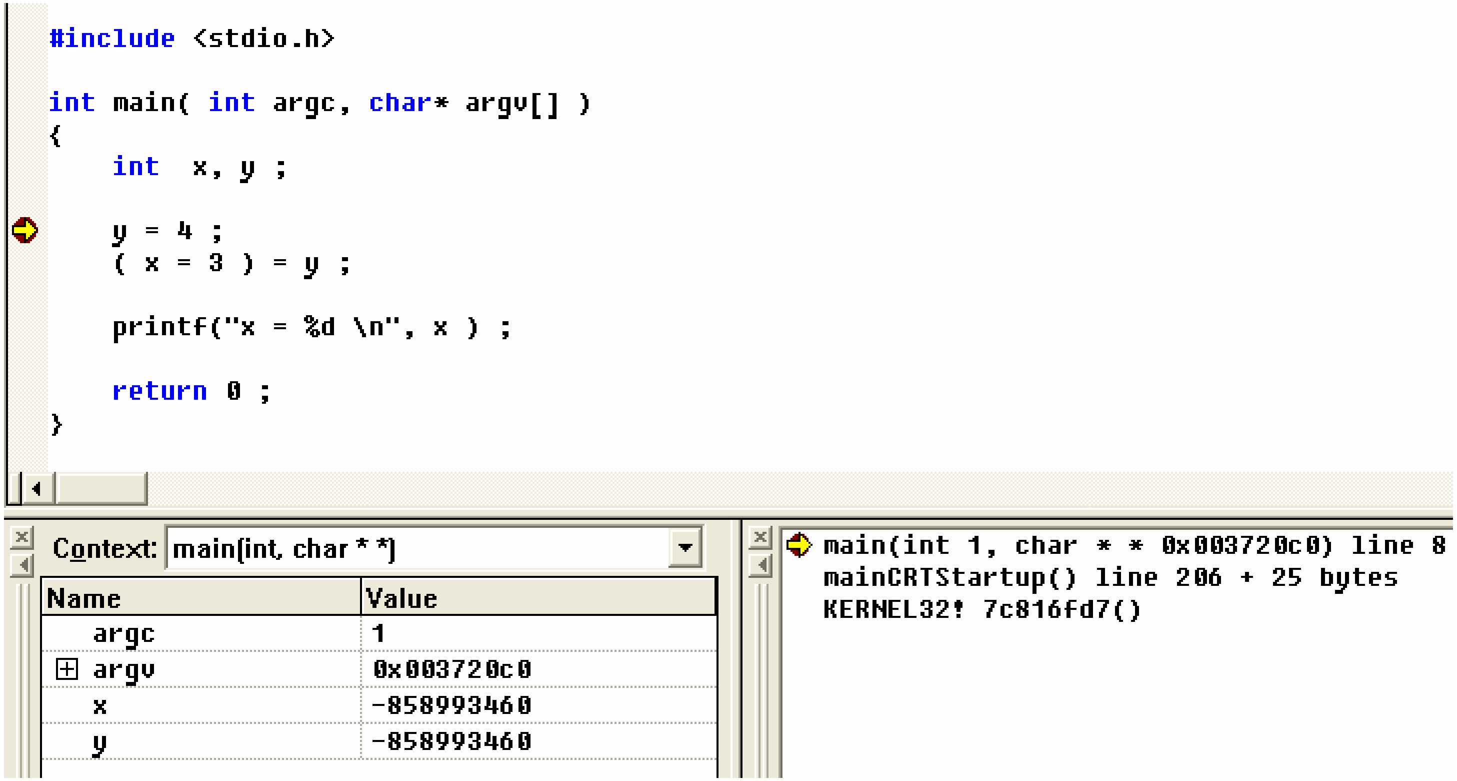


```
#include <stdio.h>


int main( int argc, char* argv[] )
{
    int  x, y ;

    y = 4 ;
    ( x = 3 ) = y ;

    printf("x = %d \n", x ) ;

    return 0 ;
}
```

Question : is expression “(x=3) = y” valid?

if so, what is the result?

中斷點 →

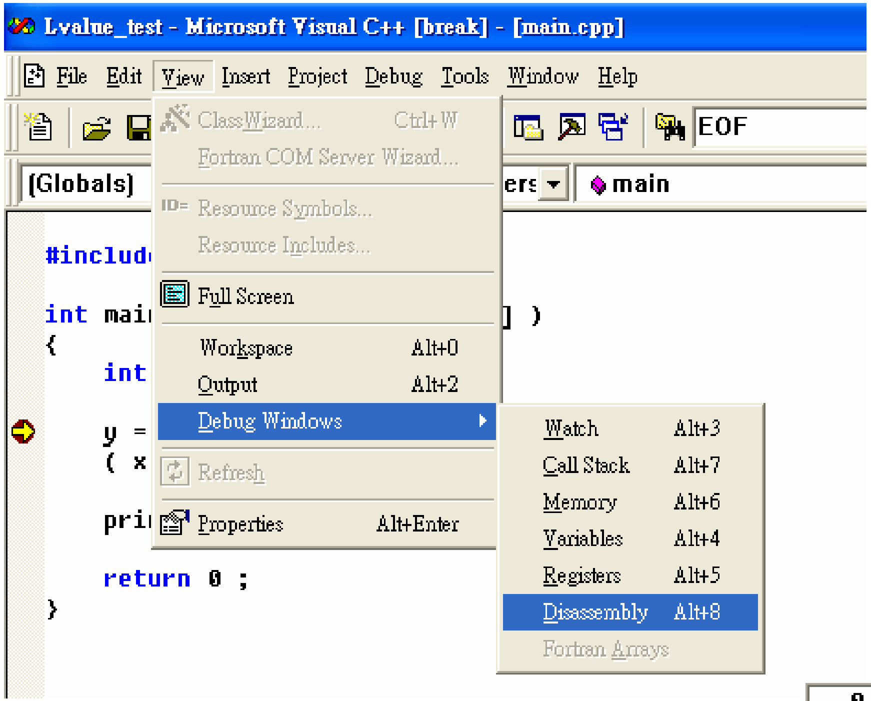


```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int  x, y ;

    y = 4 ;
    ( x = 3 ) = y ;

    printf("x = %d \n", x ) ;

    return 0 ;
}
```

Name	Value
argc	1
⊞ argv	0x003720c0
x	-858993460
y	-858993460

```
main(int 1, char * * 0x003720c0) line 8
mainCRTStartup() line 206 + 25 bytes
KERNEL32! 7c816fd7()
```

Example 6: L-value test [2]

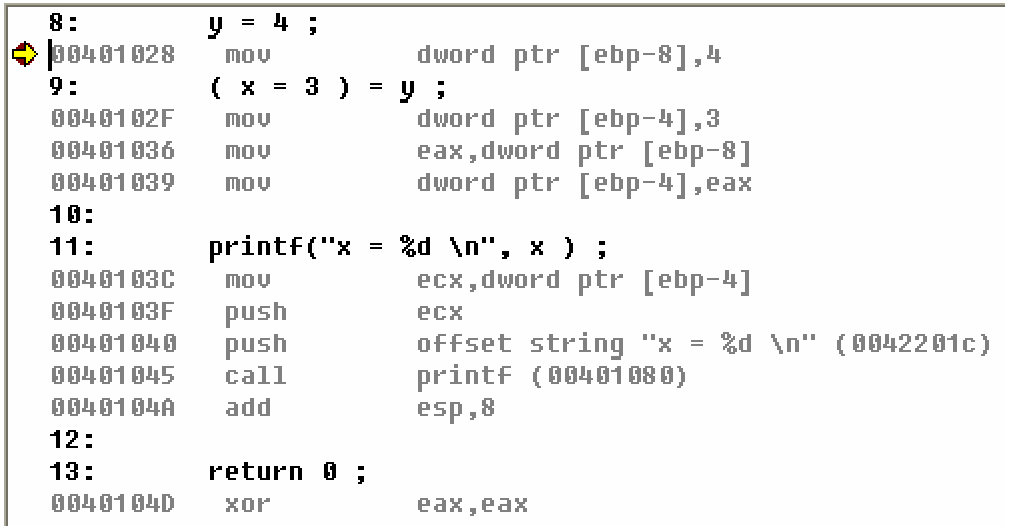


```
#includ
int mai                                 ] )
{
    int
    y =
    ( x
    pri
    return 0 ;
}
```

1. show assembly code

View → Debug Window → Disassembly

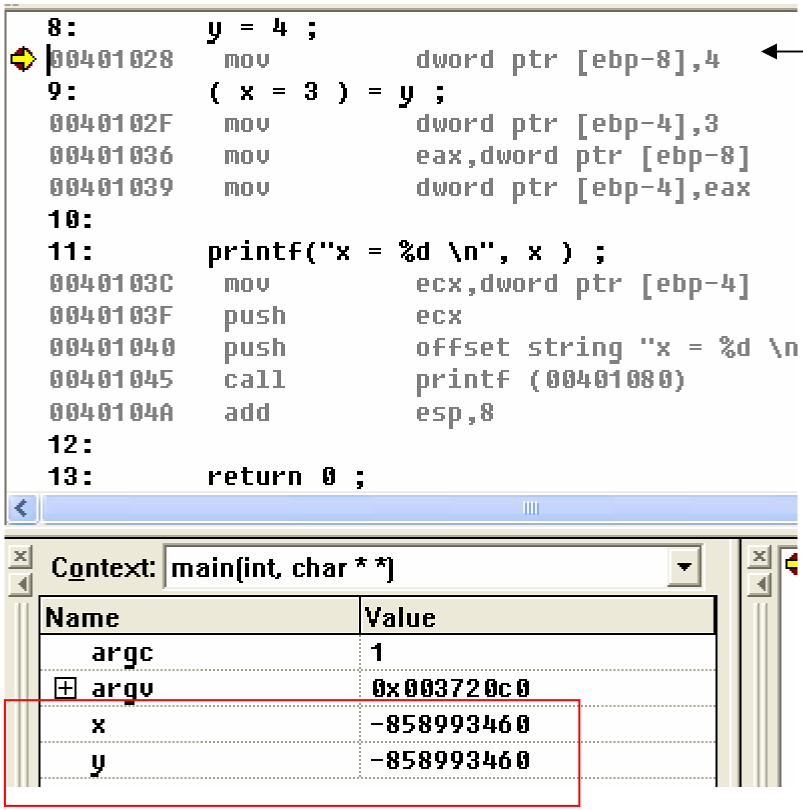
2. Assembly code (組合語言) →



```
8:          y = 4 ;
00401028    mov         dword ptr [ebp-8],4
9:          ( x = 3 ) = y ;
0040102F    mov         dword ptr [ebp-4],3
00401036    mov         eax,dword ptr [ebp-8]
00401039    mov         dword ptr [ebp-4],eax
10:
11:         printf("x = %d \n", x ) ;
0040103C    mov         ecx,dword ptr [ebp-4]
0040103F    push        ecx
00401040    push        offset string "x = %d \n" (0042201c)
00401045    call        printf (00401080)
0040104A    add         esp,8
12:
13:         return 0 ;
0040104D    xor         eax,eax
```

Example 6: L-value test [3]

1. 按 F10 執行 **y = 4**



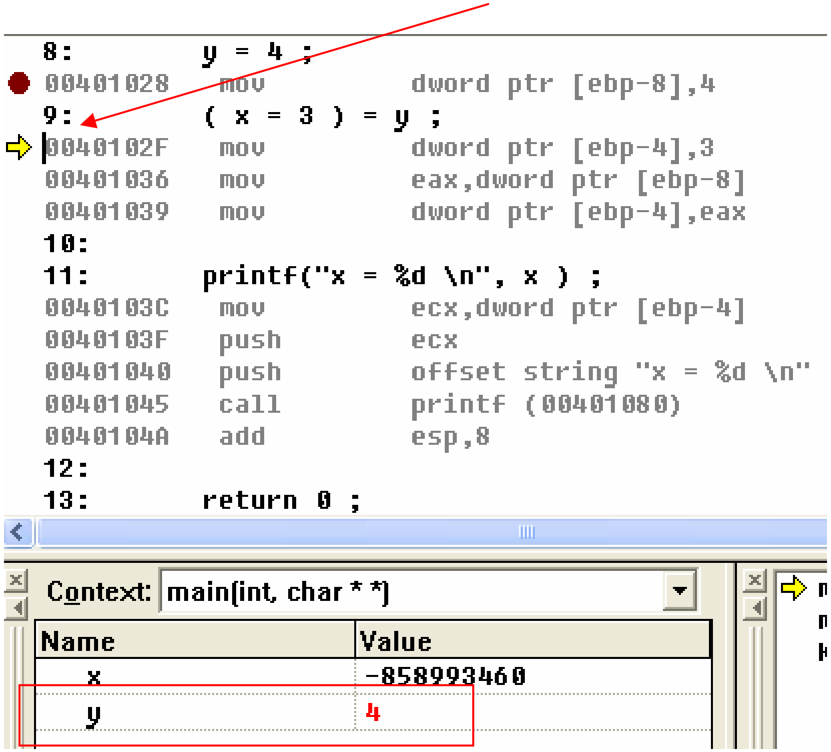
```
8:        y = 4 ;
00401028   mov         dword ptr [ebp-8],4
9:        ( x = 3 ) = y ;
0040102F   mov         dword ptr [ebp-4],3
00401036   mov         eax,dword ptr [ebp-8]
00401039   mov         dword ptr [ebp-4],eax
10:
11:       printf("x = %d \n", x ) ;
0040103C   mov         ecx,dword ptr [ebp-4]
0040103F   push        ecx
00401040   push        offset string "x = %d \n
00401045   call        printf (00401080)
0040104A   add         esp,8
12:
13:       return 0 ;
```

Context: main(int, char * *)

Name	Value
argc	1
⊞ argv	0x003720c0
x	-858993460
y	-858993460

x and y are meaningless

3. 按 F10 執行 **x = 3**



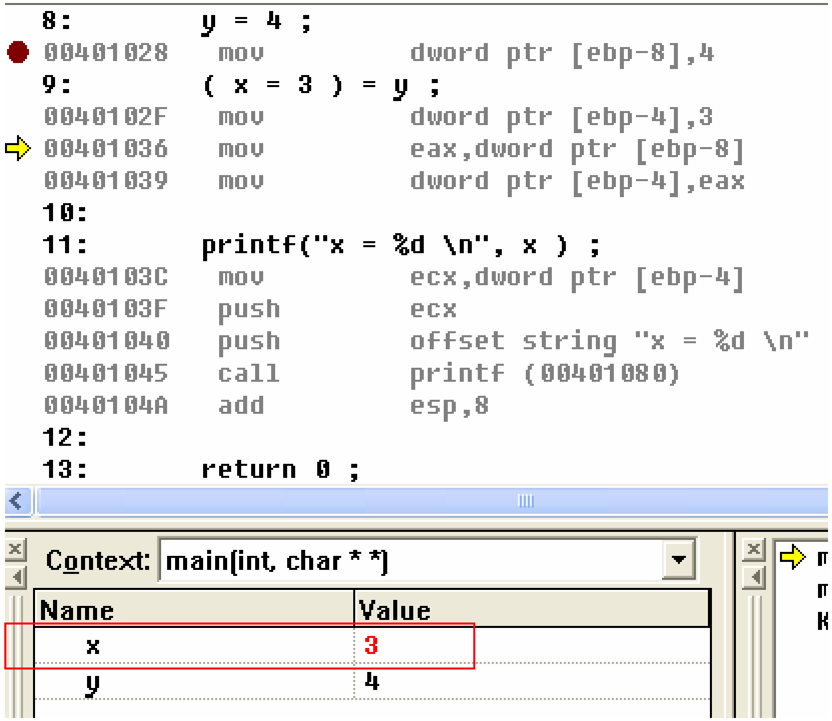
```
8:        y = 4 ;
00401028   mov         dword ptr [ebp-8],4
9:        ( x = 3 ) = y ;
0040102F   mov         dword ptr [ebp-4],3
00401036   mov         eax,dword ptr [ebp-8]
00401039   mov         dword ptr [ebp-4],eax
10:
11:       printf("x = %d \n", x ) ;
0040103C   mov         ecx,dword ptr [ebp-4]
0040103F   push        ecx
00401040   push        offset string "x = %d \n"
00401045   call        printf (00401080)
0040104A   add         esp,8
12:
13:       return 0 ;
```

Context: main(int, char * *)

Name	Value
x	-858993460
y	4

2. y = 4 is executed

Example 6: L-value test   [4]



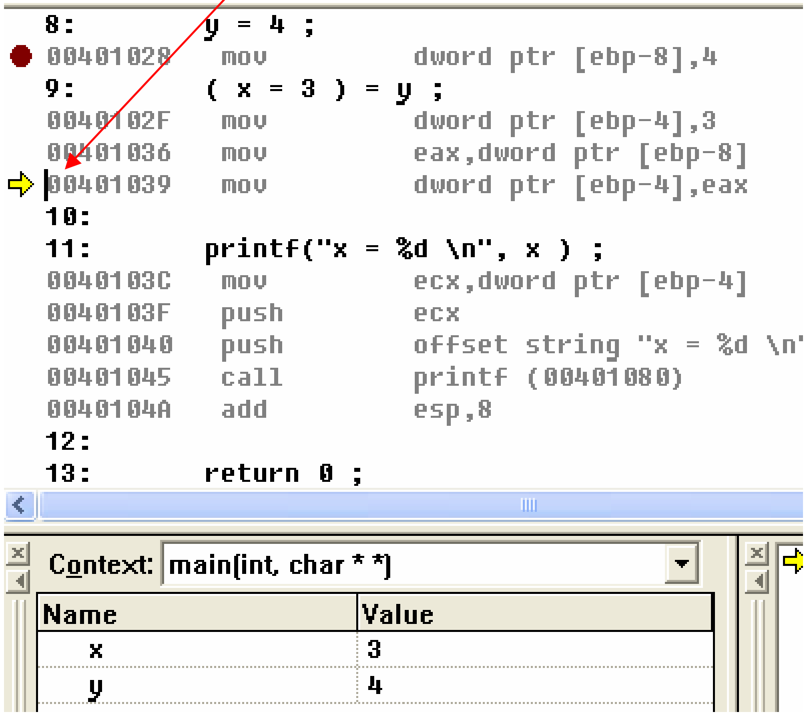
```
8:            y = 4 ;
00401028   mov          dword ptr [ebp-8],4
9:            ( x = 3 ) = y ;
0040102F   mov          dword ptr [ebp-4],3
00401036   mov          eax,dword ptr [ebp-8]
00401039   mov          dword ptr [ebp-4],eax
10:
11:           printf("x = %d \n", x ) ;
0040103C   mov          ecx,dword ptr [ebp-4]
0040103F   push         ecx
00401040   push         offset string "x = %d \n"
00401045   call         printf (00401080)
0040104A   add          esp,8
12:
13:           return 0 ;
```

Context: main(int, char * *)

Name	Value
x	3
y	4

x = 3 is executed

按 F10 執行 **x = y**

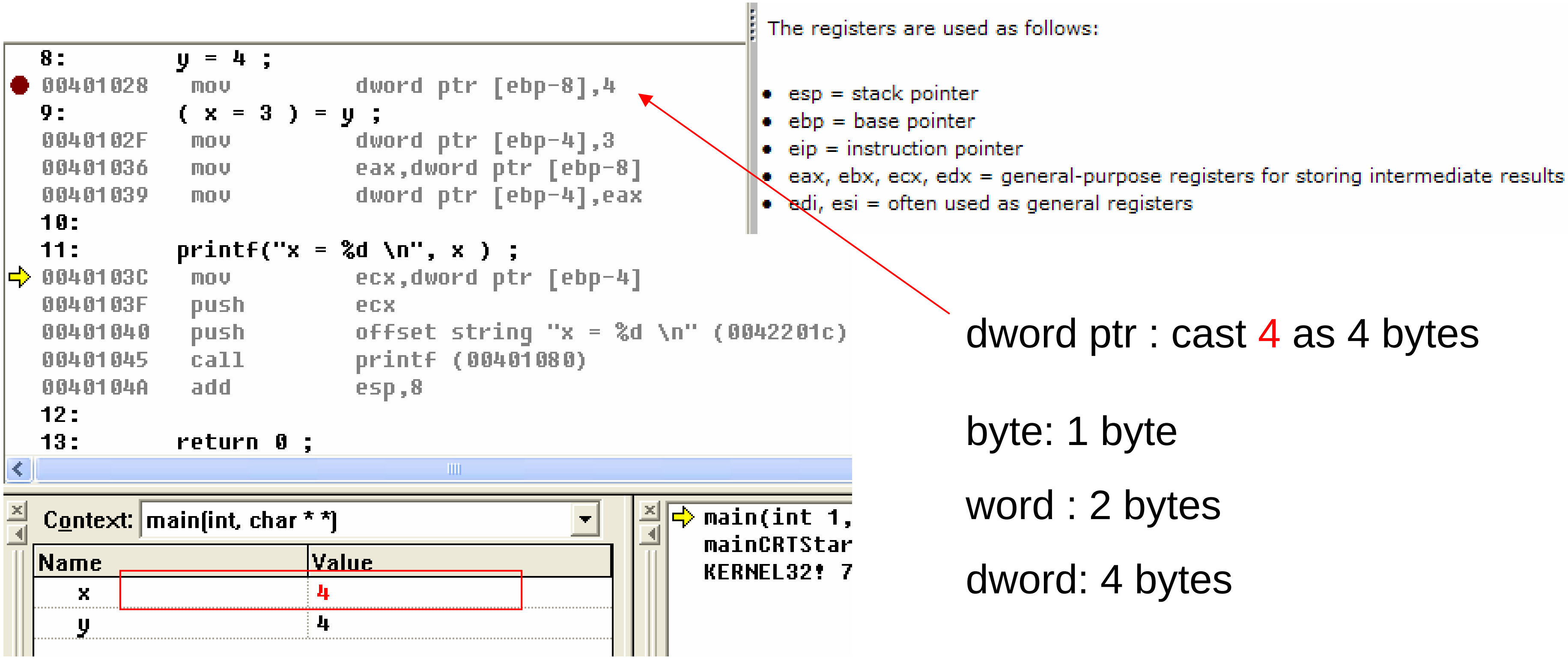


```
8:            y = 4 ;
00401028   mov          dword ptr [ebp-8],4
9:            ( x = 3 ) = y ;
0040102F   mov          dword ptr [ebp-4],3
00401036   mov          eax,dword ptr [ebp-8]
00401039   mov          dword ptr [ebp-4],eax
10:
11:           printf("x = %d \n", x ) ;
0040103C   mov          ecx,dword ptr [ebp-4]
0040103F   push         ecx
00401040   push         offset string "x = %d \n'
00401045   call         printf (00401080)
0040104A   add          esp,8
12:
13:           return 0 ;
```

Context: main(int, char * *)

Name	Value
x	3
y	4

Example 6: L-value test [5]



```
8:         y = 4 ;
00401028   mov         dword ptr [ebp-8],4
9:         ( x = 3 ) = y ;
0040102F   mov         dword ptr [ebp-4],3
00401036   mov         eax,dword ptr [ebp-8]
00401039   mov         dword ptr [ebp-4],eax
10:
11:        printf("x = %d \n", x ) ;
0040103C   mov         ecx,dword ptr [ebp-4]
0040103F   push        ecx
00401040   push        offset string "x = %d \n" (0042201c)
00401045   call        printf (00401080)
0040104A   add         esp,8
12:
13:        return 0 ;
```

Context: main(int, char * *)

Name	Value
x	4
y	4

```
main(int 1,
mainCRTStar
KERNEL32! 7
```

The registers are used as follows:

- esp = stack pointer
- ebp = base pointer
- eip = instruction pointer
- eax, ebx, ecx, edx = general-purpose registers for storing intermediate results
- edi, esi = often used as general registers

dword ptr : cast 4 as 4 bytes

byte: 1 byte

word : 2 bytes

dword: 4 bytes

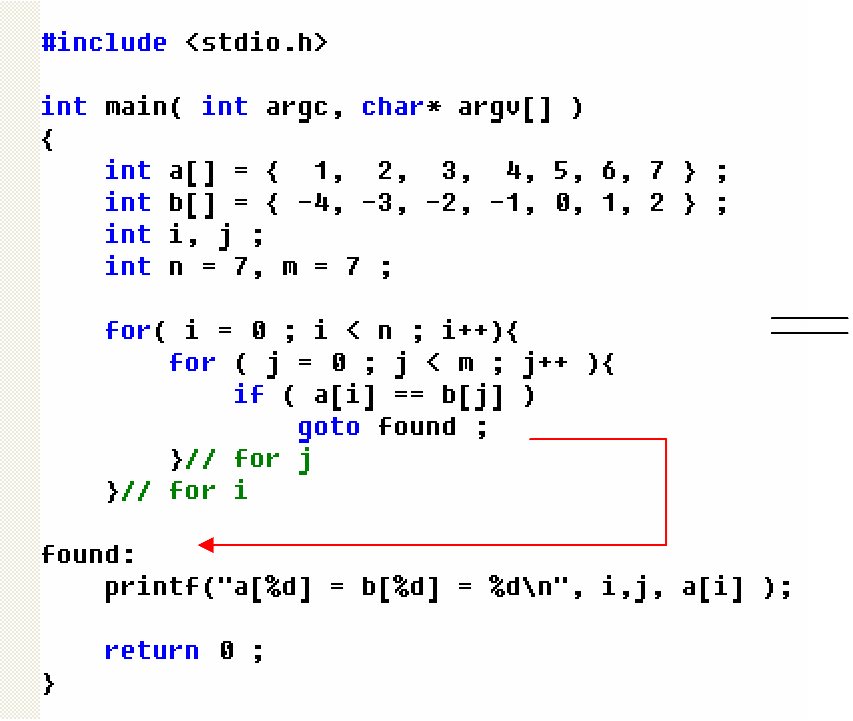
x = y is executed

OutLine

- expression and statement
- selection statement
- iteration statement
- Visual Studio debugger
- goto and labels

Example 7: goto and labels

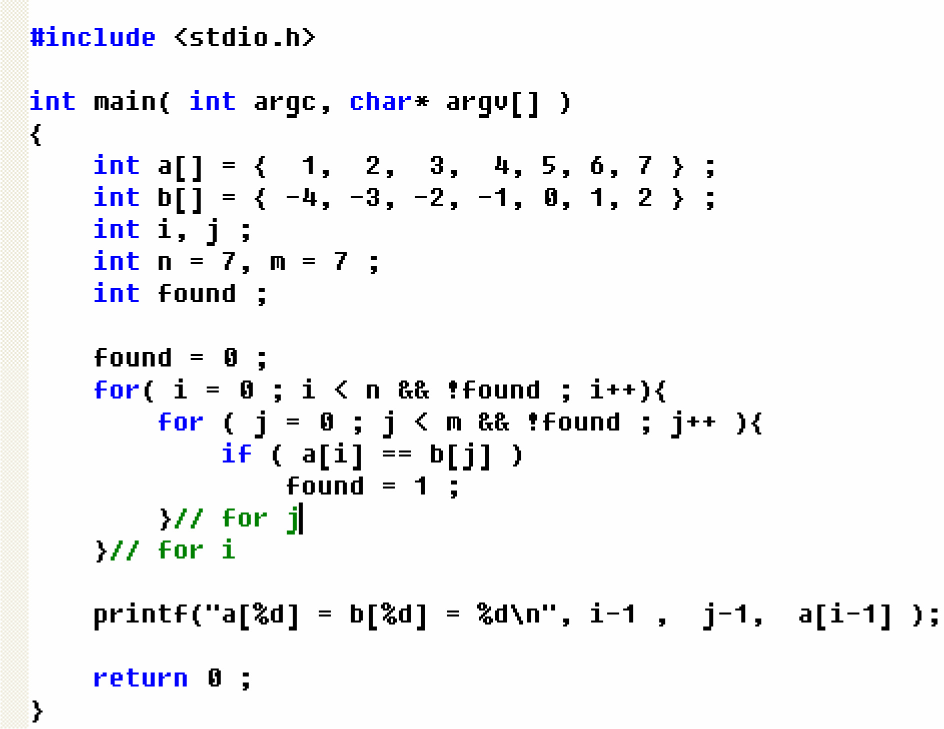
With goto



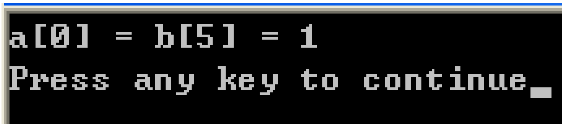
```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int a[] = {  1,  2,  3,  4, 5, 6, 7 } ;
    int b[] = { -4, -3, -2, -1, 0, 1, 2 } ;
    int i, j ;
    int n = 7, m = 7 ;

    for( i = 0 ; i < n ; i++){
        for ( j = 0 ; j < m ; j++ ){
            if ( a[i] == b[j] )
                goto found ;
        }// for j
    }// for i

found:
    printf("a[%d] = b[%d] = %d\n", i,j, a[i] );

    return 0 ;
}
```

=

Without goto



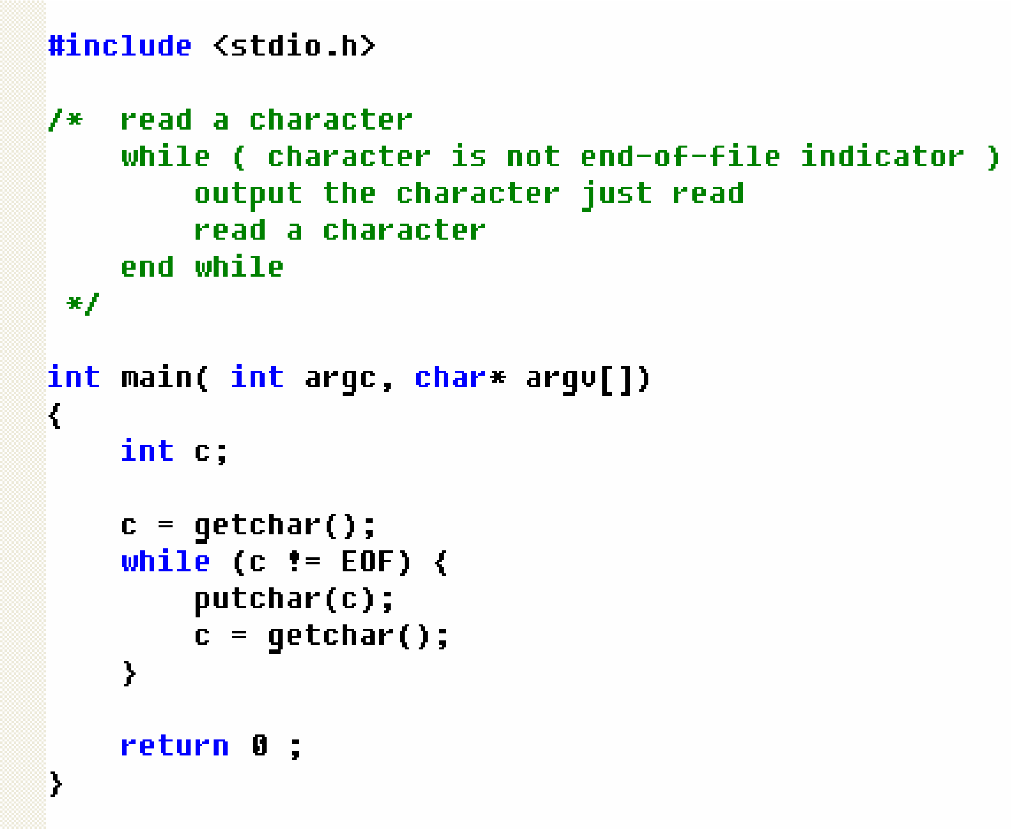
```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int a[] = {  1,  2,  3,  4, 5, 6, 7 } ;
    int b[] = { -4, -3, -2, -1, 0, 1, 2 } ;
    int i, j ;
    int n = 7, m = 7 ;
    int found ;

    found = 0 ;
    for( i = 0 ; i < n && !found ; i++){
        for ( j = 0 ; j < m && !found ; j++ ){
            if ( a[i] == b[j] )
                found = 1 ;
        }// for j
    }// for i

    printf("a[%d] = b[%d] = %d\n", i-1 ,  j-1,  a[i-1] );

    return 0 ;
}
```



```
a[0] = b[5] = 1
Press any key to continue_
```

Don't use goto

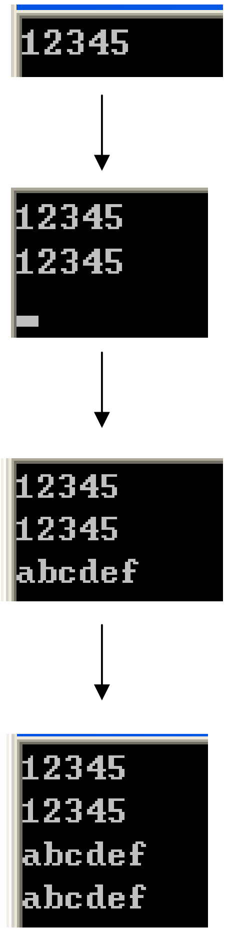
Exercise : EOF versus stdin [1]



```c
#include <stdio.h>

/*  read a character
    while ( character is not end-of-file indicator )
        output the character just read
        read a character
    end while
 */

int main( int argc, char* argv[])
{
    int c;

    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }

    return 0 ;
}
```

int ***getchar*** (void) : get one character from standard input, see page 247

Result under Visual C



```
12345
```

press enter

```
12345
12345
```

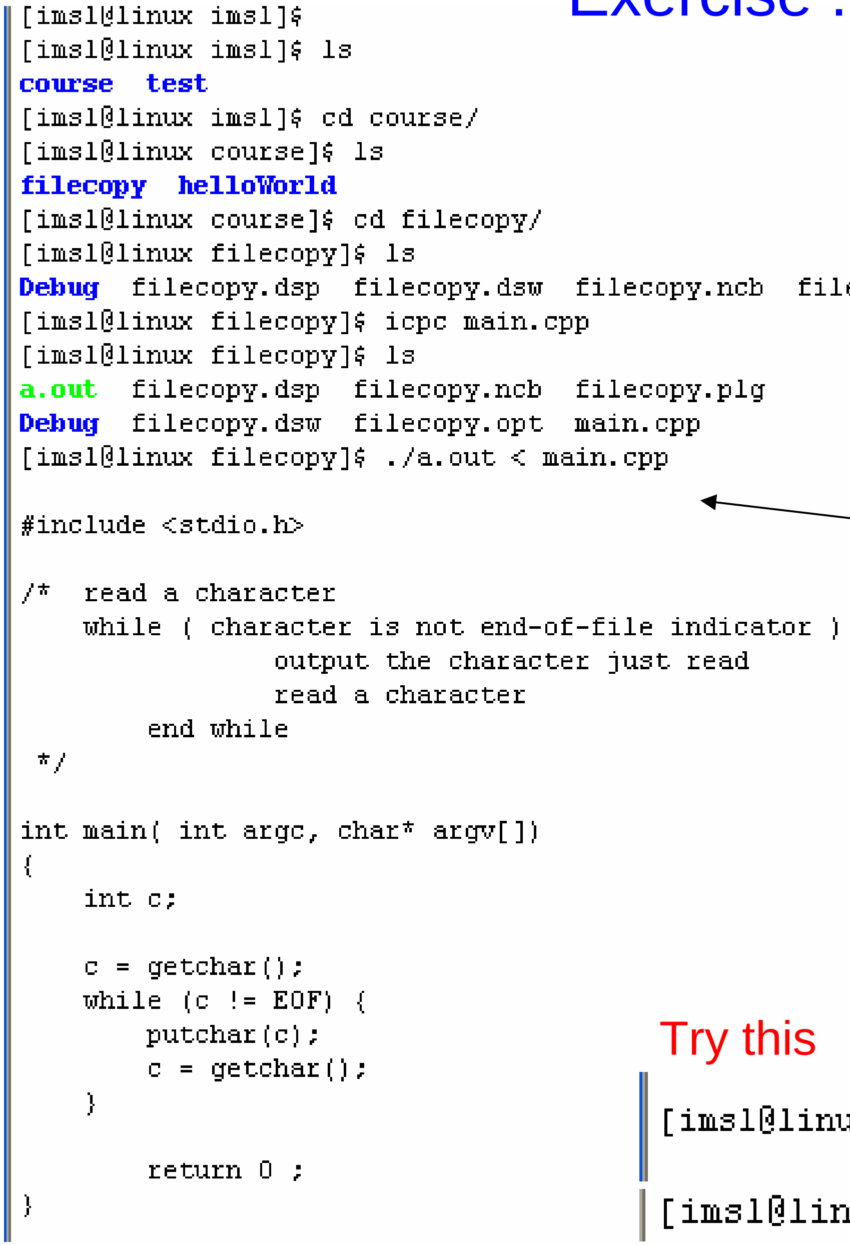
```
12345
12345
abcdef
```

press enter

```
12345
12345
abcdef
abcdef
```

Question: why enter infinite loop?

Exercise : EOF versus stdin [2]

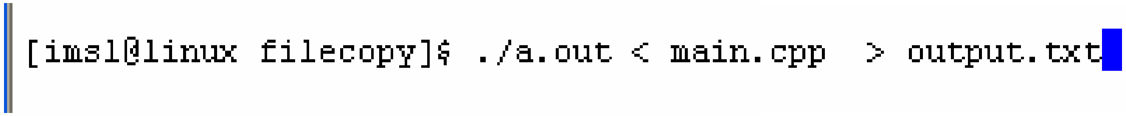


```
[imsl@linux imsl]$
[imsl@linux imsl]$ ls
course  test
[imsl@linux imsl]$ cd course/
[imsl@linux course]$ ls
filecopy  helloWorld
[imsl@linux course]$ cd filecopy/
[imsl@linux filecopy]$ ls
Debug  filecopy.dsp  filecopy.dsw  filecopy.ncb  filecopy.opt  filecopy.plg  main.cpp
[imsl@linux filecopy]$ icpc main.cpp
[imsl@linux filecopy]$ ls
a.out  filecopy.dsp  filecopy.ncb  filecopy.plg
Debug  filecopy.dsw  filecopy.opt  main.cpp
[imsl@linux filecopy]$ ./a.out < main.cpp

#include <stdio.h>

/*  read a character
    while ( character is not end-of-file indicator )
                output the character just read
                read a character
        end while
 */

int main( int argc, char* argv[])
{
    int c;

    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }

        return 0 ;
}
```

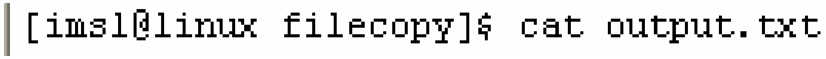
Feed file **main.cpp** into **a.out**

這是UNIX 的檔案導向功能

Try this



```
[imsl@linux filecopy]$ ./a.out < main.cpp  > output.txt
```



```
[imsl@linux filecopy]$ cat output.txt
```