Chapter 7 Input and Output

Speaker: Lung-Sheng Chien

Re-direct and pipe in Unix system

- A file may be substituted for the keyboard by using < convention for input redirection
  ./a.out < article.txt
  the string "< article.txt" is not included in the command-line arguments in *argv*.

- Output can be redirected into a file with > *filename*
  ./a.out > output.txt
  write standard output to file output.txt

- Input switching is also invisible if the input comes from another program via a pipe mechanism
  man icpc | more
  put the standard output of "*man icpc*" into standard input of *more*

OutLine

- Format of printf
- Variable-length argument lists
- Formatted input – scanf
- File access
- Command execution

Formatted output – printf [1]

int ***printf*** (char ****format***, **arg1**, ***arg2***, …)

```
#include <stdio.h>

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

    printf("%-15s:%s: \n"       , ":%s:"      , word);
    printf("%-15s:%10s: \n"     , ":%10s:"    , word);
    printf("%-15s:%.10s: \n"    , ":%.10s:"   , word);
    printf("%-15s:%-10s: \n"    , ":%-10s:"   , word);
    printf("%-15s:%.15s: \n"    , ":%.15s:"   , word);
    printf("%-15s:%-15s: \n"    , ":%-15s:"   , word);
    printf("%-15s:%15.10s: \n"  , ":%15.10s:" , word);
    printf("%-15s:%-15.10s: \n" , ":%-15.10s:", word);

    return 0 ;
}
```

```
"F:\COURSE\2008SUMMER\C_LANG\EXAMP
:%s:            :hello, world:
:%10s:          :hello, world:
:%.10s:         :hello, wor:
:%-10s:         :hello, world:
:%.15s:         :hello, world:
:%-15s:         :hello, world   :
:%15.10s:       :     hello, wor:
:%-15.10s:      :hello, wor     :
Press any key to continue
```

format

%s

1	2	3	4	5	6	7	8	9	10	11	12
h	*e*	*l*	*l*	*o*	,		*w*	*o*	*r*	*l*	*d*

%10s

1	2	3	4	5	6	7	8	9	10	11	12
h	*e*	*l*	*l*	*o*	,		*w*	*o*	*r*	*l*	*d*

Formatted output – printf [2]

Format specification (from MSDN library)

%[flags] [width] [.precision] [{h | l | ll | I | I32 | I64}]type

- ***type***
  - required character that determines whether the associated argument is interpreted as a character, a string or a number.

- ***flags***
  - Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes.

- ***width***
  - Optional number that specifies the minimum number of characters output.

- ***precision***
  - Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

Format specification: *type*

Character	Type	Output format
c	**int** or **wint_t**	When used with **printf** functions, specifies a single-byte character; when used with **wprintf** functions, specifies a wide character.
d	**int**	Signed decimal integer.
o	**int**	Unsigned octal integer.
u	**int**	Unsigned decimal integer.
x	**int**	Unsigned hexadecimal integer, using "abcdef."
e	**double**	Signed value having the form [–]*d.dddd* **e** [*sign*]*dd[d]* where *d* is a single decimal digit, *dddd* is one or more decimal digits, *dd[d]* is two or three decimal digits depending on the output format and size of the exponent, and *sign* is + or –.
E	**double**	Identical to the **e** format except that **E** rather than **e** introduces the exponent.
f	**double**	Signed value having the form [–]*dddd.dddd*, where *dddd* is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
p	Pointer to **void**	Prints the address of the argument in hexadecimal digits.
s	String	When used with **printf** functions, specifies a single-byte–character string; when used with **wprintf** functions, specifies a wide-character string. Characters are printed up to the first null character or until the *precision* value is reached.

Format specification: *flag*

Flag	Meaning	Default
–	Left align the result within the given field width.	Right align.
+	Prefix the output value with a sign (+ or –) if the output value is of a signed type.	Sign appears only for negative signed values (–).
0	If *width* is prefixed with **0**, zeros are added until the minimum width is reached. If **0** and – appear, the **0** is ignored. If **0** is specified with an integer format (**i**, **u**, **x**, **X**, **o**, **d**) and a precision specification is also present (for example, `%04.d`), the **0** is ignored.	No padding.
blank (' ')	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.
#	When used with the **o**, **x**, or **X** format, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively.	No blank appears.
	When used with the **e**, **E**, **f**, **a** or **A** format, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.
	When used with the **g** or **G** format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Ignored when used with **c**, **d**, **i**, **u**, or **s**.	Decimal point appears only if digits follow it. Trailing zeros are truncated.

Format specification: *width*

- The *width* argument is a nonnegative decimal integer controlling the minimum number of characters printed.
- If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values — depending on whether the – *flag* (for left alignment) is specified.
- If the number of characters in the output value is greater than the specified width, or if *width* is not given, all characters of the value are printed.
- If the *width* specification is an asterisk (*****), an **int** argument from the argument list supplies the value (see page 154). This case is rare.

Format specification: *precision*

How Precision Values Affect Type

c, C	The precision has no effect.	Character is printed.
d, i, u, o, x, X	The precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than *precision*, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds *precision*.	Default precision is 1.
e, E	The precision specifies the number of digits to be printed after the decimal point. The last printed digit is rounded.	Default precision is 6; if *precision* is 0 or the period (.) appears without a number following it, no decimal point is printed.
f	The precision value specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.	Default precision is 6; if *precision* is 0, or if the period (.) appears without a number following it, no decimal point is printed.
g, G	The precision specifies the maximum number of significant digits printed.	Six significant digits are printed, with any trailing zeros truncated.
s, S	The precision specifies the maximum number of characters to be printed. Characters in excess of *precision* are not printed.	Characters are printed until a null character is encountered.

Format spec : *example 1*

%[flags] [width] [.precision] [{h | l | ll | I | I32 | I64}]type

	1	2	3	4	5	6	7	8	9	10	11	12
%10s	*h*	*e*	*l*	*l*	*o*	*,*		*w*	*o*	*r*	*l*	*d*

flag = empty ***width*** = 10 ***precision*** = empty ***type*** = string

length(“hello, world”) = 12 > width = 10

Rule : If the number of characters in the output value is greater than the specified width, all characters of the value are printed.

print last two characters also

1	2	3	4	5	6	7	8	9	10	11	12
h	*e*	*l*	*l*	*o*	*,*		*w*	*o*	*r*	*l*	*d*

Format spec : *example 2*

%[flags] [width] [.precision] [{h | l | ll | I | I32 | I64}]type

%-15.10s

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h	*e*	*l*	*l*	*o*	*,*		*w*	*o*	*r*					

flag = minus

left alignment

width = 15

precision = 10

Characters in excess of ***precision*** are not printed.

type = string

width = 15

left alignment

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h	*e*	*l*	*l*	*o*	*,*		*w*	*o*	*r*	*l*	*d*			

OutLine

- Format of printf
- Variable-length argument lists
- Formatted input – scanf
- File access
- Command execution

Variable-length Argument Lists

int ***printf*** (char ****fmt***, ***...***)

The number and types of these arguments may vary

Question 1: how to walk through argument list when this list doesn't have a name and type?

Question 2: how to know number of arguments in the argument list?

```
char word[] = "hello, world" ;

printf("%-15s:%s: \n"       , ":%s:"      , word);
```

fmt ***arg1*** ***arg2***

Header file: stdarg.h

- void ***va_start*** (va_list ***arg_ptr***, ***prev_param***)
  va_start sets *arg_ptr* to the first optional argument in the list of arguments passed to the function. The argument *arg_ptr* must have **va_list** type. The argument *prev_param* is the name of the required parameter immediately preceding the first optional argument in the argument list.

- ***type va_arg***(va_list ***arg_ptr***, ***type***)
  va_arg retrieves a value of type from the location given by *arg_ptr* and increments *arg_ptr* to point to the next argument in the list, using the size of type to determine where the next argument starts.

- void ***va_end***(va_list ***arg_list***)

  va_end resets the pointer to **NULL.**

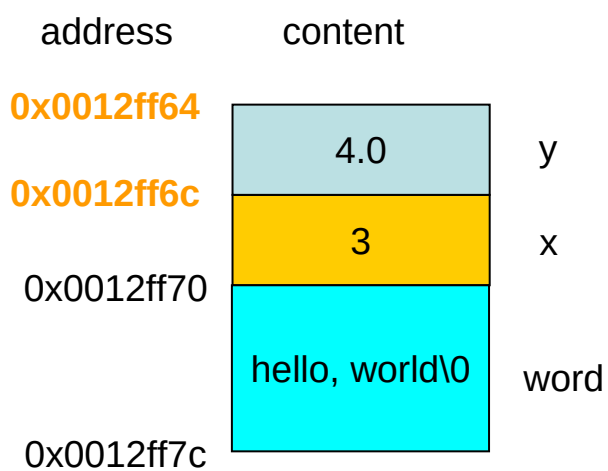
Example: min-printf

minprintf.cpp

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

// minprintf: miniml printf with variable argument list
void minprintf( char *fmt, ... )
{
    va_list ap ; // points to each unnamed arg in turn
    char *p, *sval ;
    int ival ;
    double dval ;

1   va_start(ap, fmt); // make ap point to 1st unamed arg
    for ( p = fmt ; *p ; p++ ){
        if ( '%' != *p ){
            putchar(*p) ;
            continue ;
        }
        switch( *++p ){
        case 'd' :
2           ival = va_arg(ap, int) ;
            printf("%d", ival) ;
            break ;
        case 'f' :
2           dval = va_arg(ap, double) ;
            printf("%f", dval) ;
            break ;
        case 's' :          2
            for ( sval = va_arg(ap, char*) ; *sval ; sval++){
                putchar( *sval ) ;
            }
            break ;
        default:
            putchar( *p ) ;
            break ;
        }

    }// for each character *p
3   va_end( ap ) ; // clean up when done
}
```

main.cpp

```
#include <stdio.h>

void minprintf( char *fmt, ... ) ;

int main( int argc, char *argv[] )
{
    char word[] = "hello, world" ;
    int      x = 3 ;
    double   y = 4.0 ;

    minprintf( "x = %d, y = %f, word = %s\n",
           x, y, word ) ;

    return 0 ;
}
```



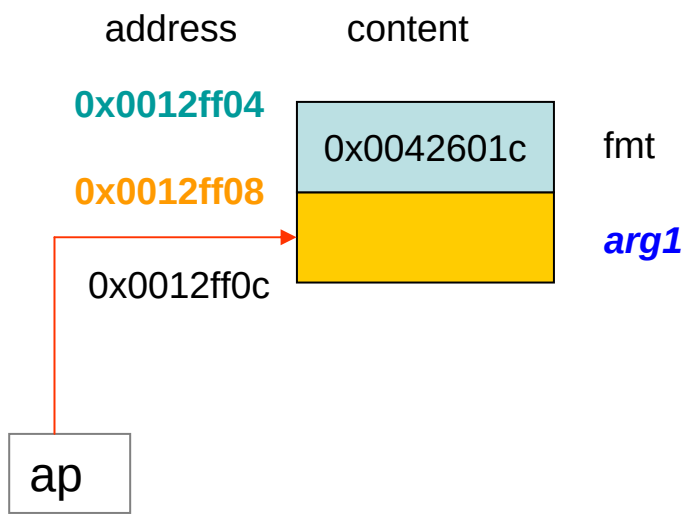
Macro va_start

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

// minprintf: minimal printf with variable argument list
void minprintf( char *fmt, ... )
{
    va_list ap ; // points to each unnamed arg in turn
    char *p, *sval ;
    int ival ;
    double dval ;

    va_start(ap, fmt); // make ap point to 1st unamed arg
    for ( p = fmt ; *p ; p++ ){
        if ( '%' != *p ){
            putchar(*p) ;
            continue ;
        }
        switch( *++p ){
```

Context: minp

Name	Value
ap	0x001 "ﾌ"
fmt	0x004 "x =

Name	Value
&fmt	0x0012ff04
	0x0042601c "x = %d, "
ap	0x0012ff08 "ﾌ"

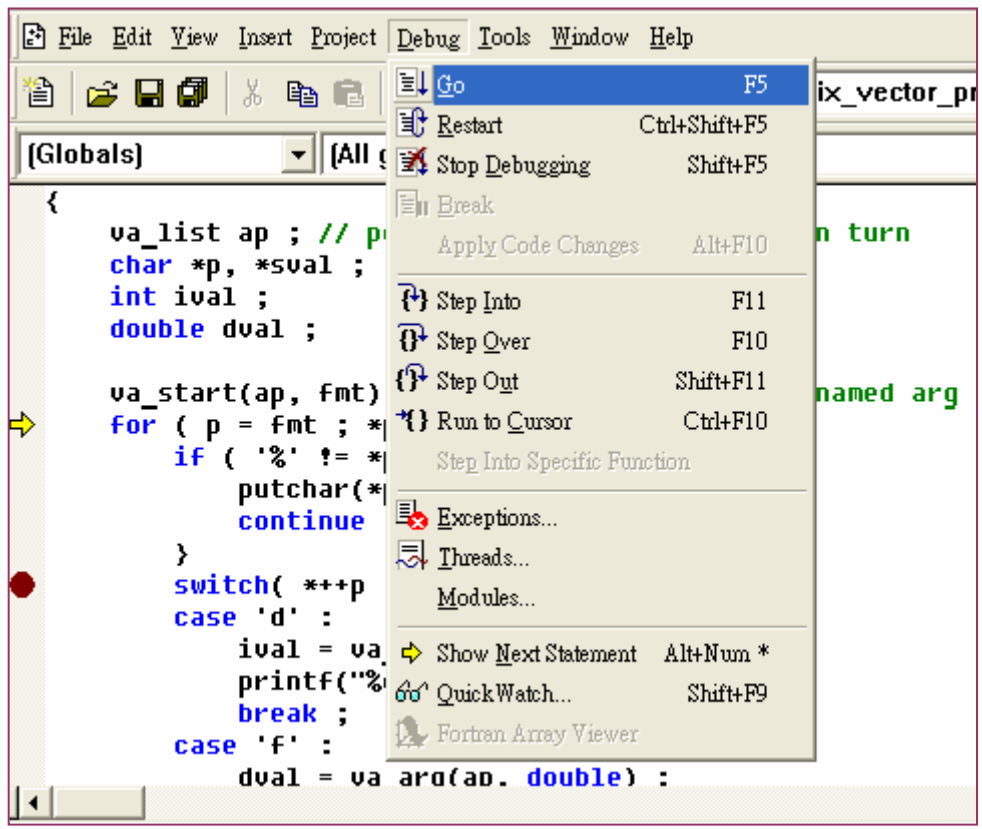


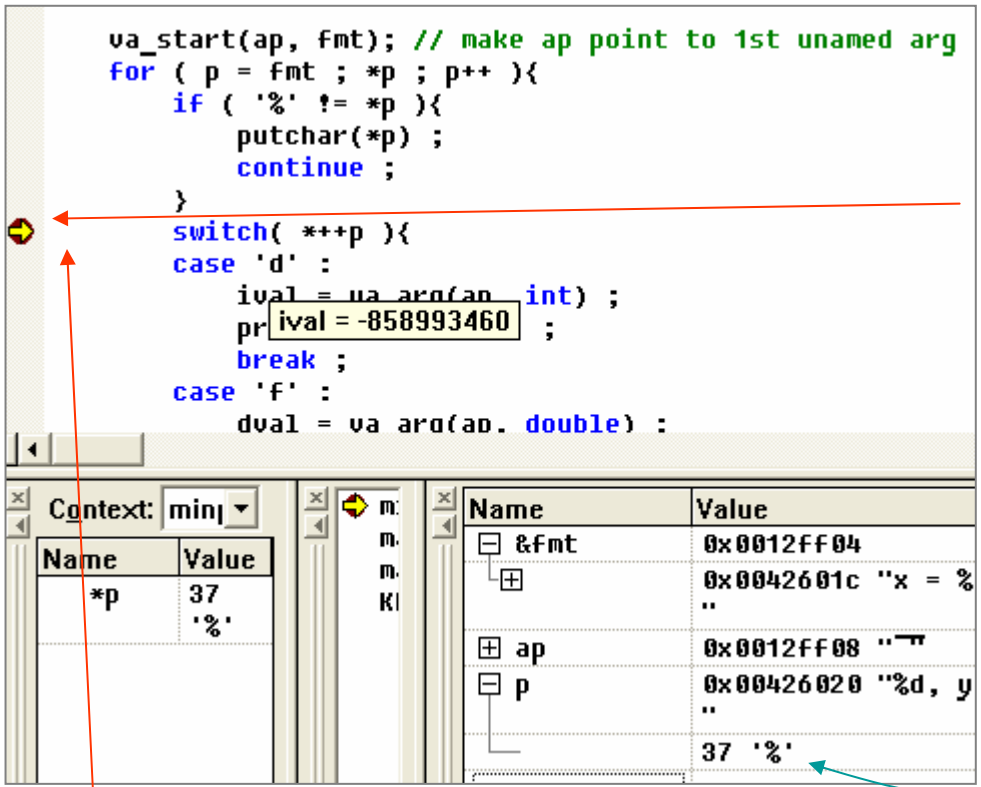
Given variable name ***fmt***, macro ***va_start*** compute next argument adjacent to ***fmt***

```
va_start(ap, fmt); // make ap point to 1
for ( p = fmt ; *p ; p++ ){
    if ( '%' != *p ){
        putchar(*p) ;
        continue ;
    }
    switch( ***p ){
    case 'd' :
        ival = va_arg(ap, int) ;
        printf("%d", ival) ;
        break ;
    case 'f' :
        dval = va_arg(ap, double) ;
```

1. 按 F9 產生一個中斷點

2. Debug → Go 至下一個中斷點

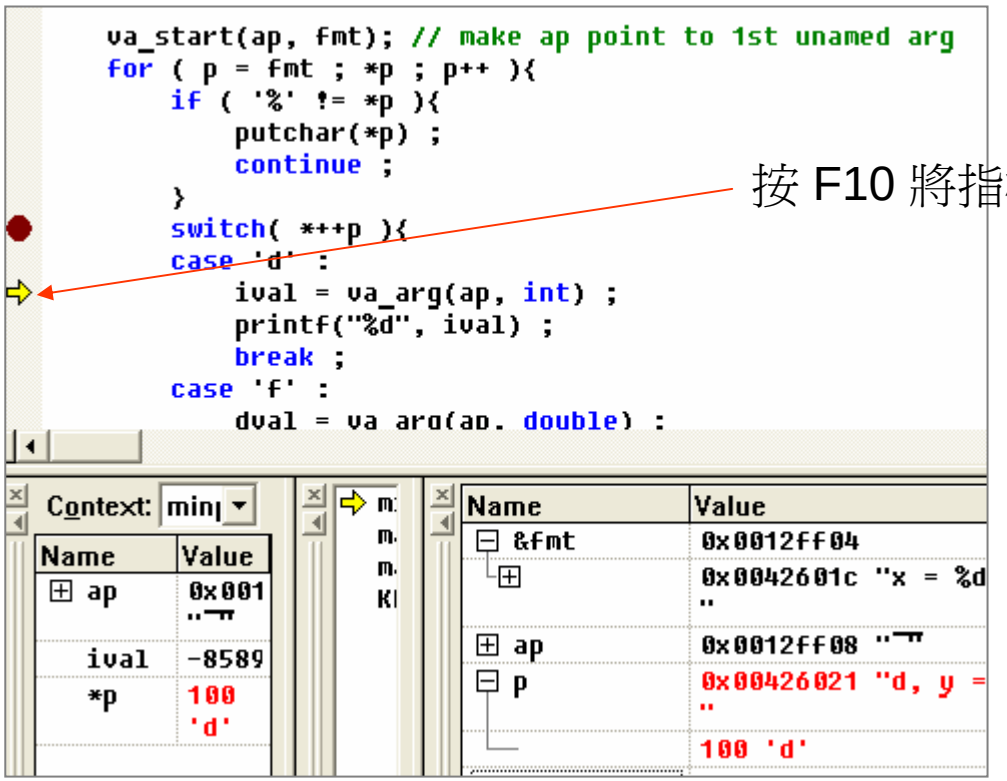




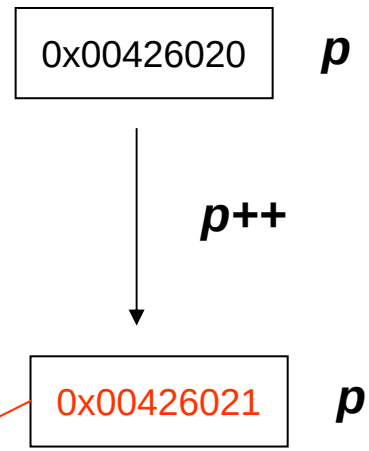
In order to reach this break-point, condition in if-else clause must be false, say ***p == '%'***

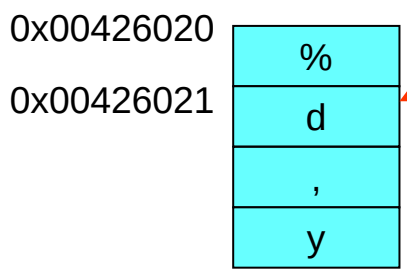
matched

按 F10 執行 ***p++,*** 爾後再執行 ****p***

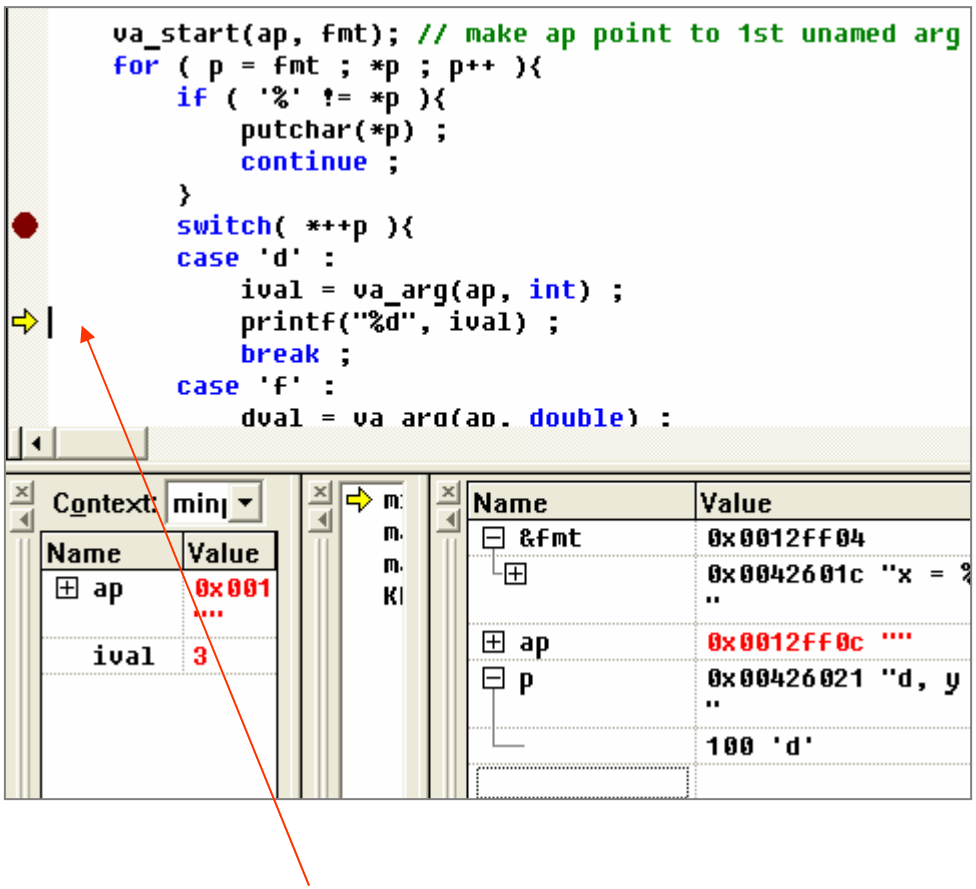


按 F10 將指標 ap 移下一個 int 的長度





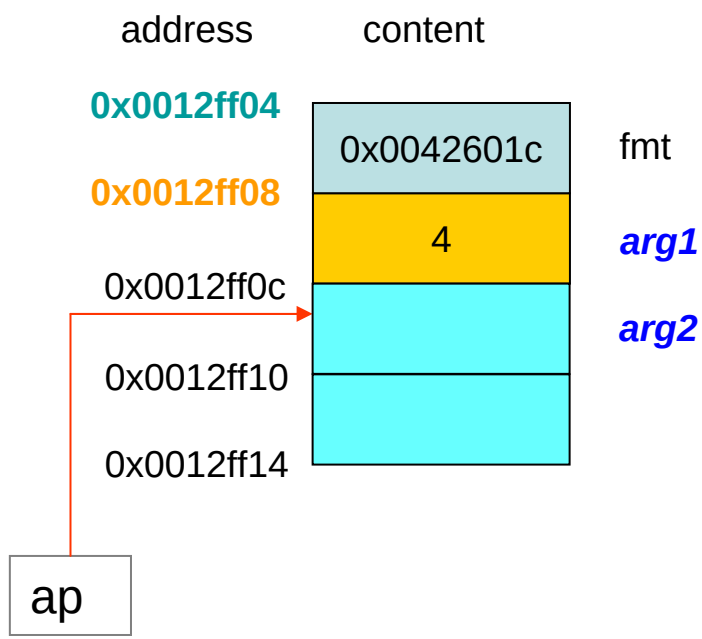
Macro va_arg

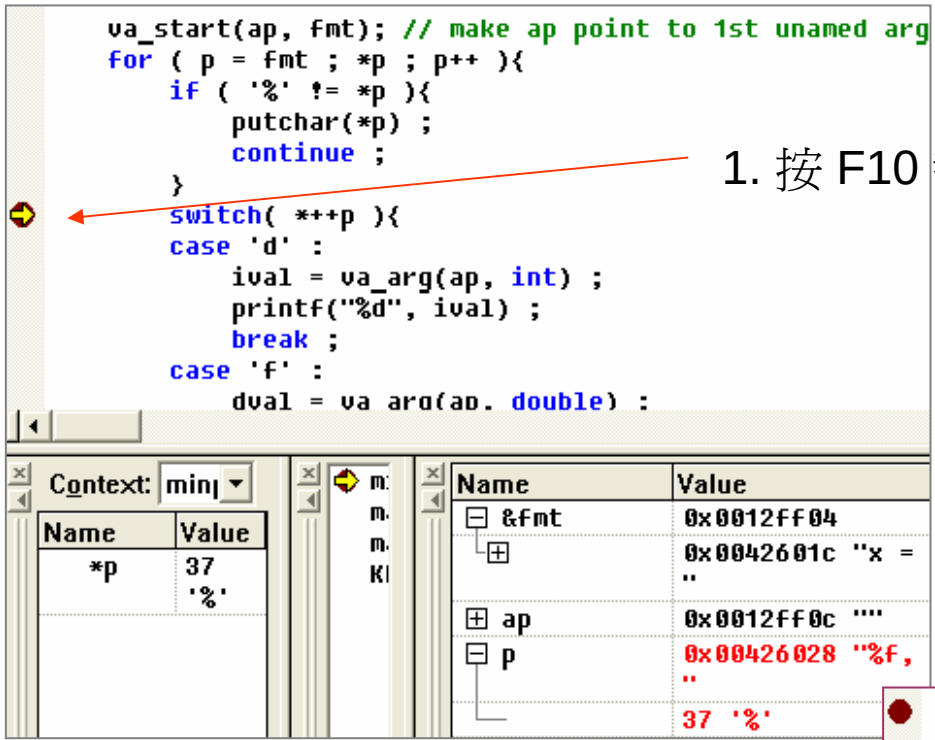


Debug → Go 至下一個中斷點

1. 以 int 型態取出 ***ap*** 所指的內容, 並存入 ***ival*** 中

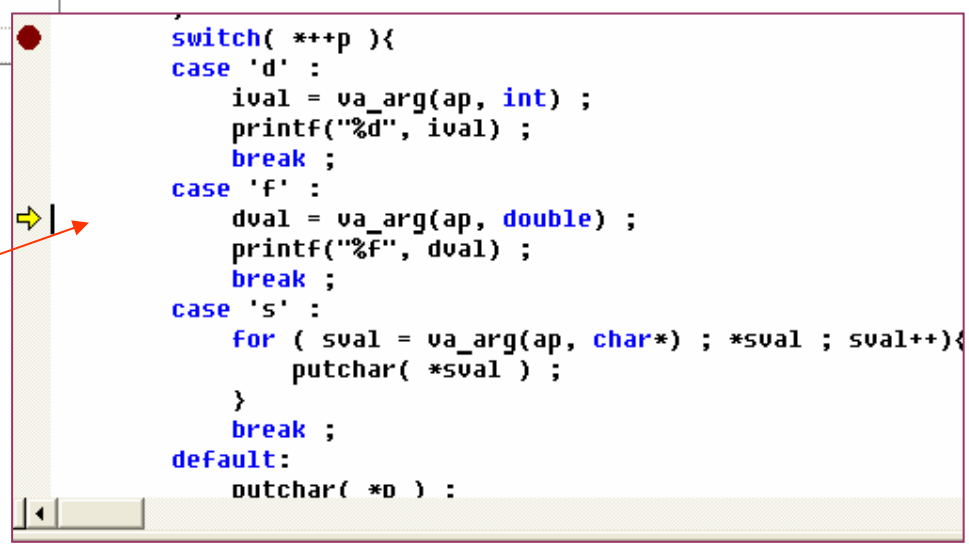
2. 移動指標 ap 一個 int 的長度, 即 ap 指向下一個 argument (***arg2***)

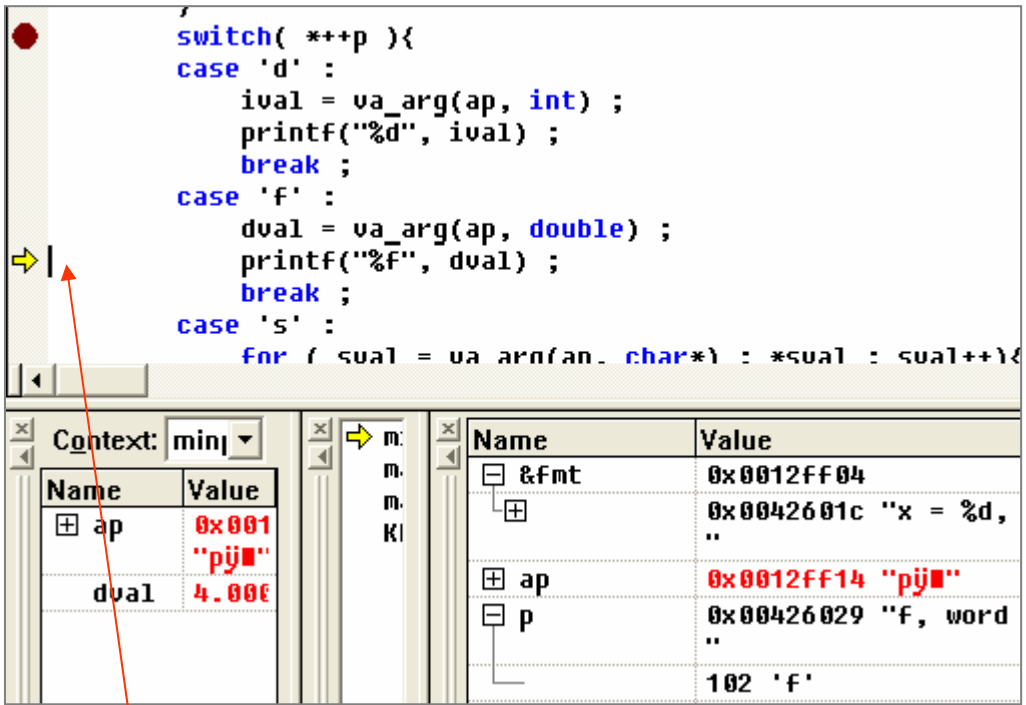




1. 按 F10 執行 ***p++,*** 爾後再執行 ****p***

2. 按 F10 將指標 ap 移下一個 double 的長度

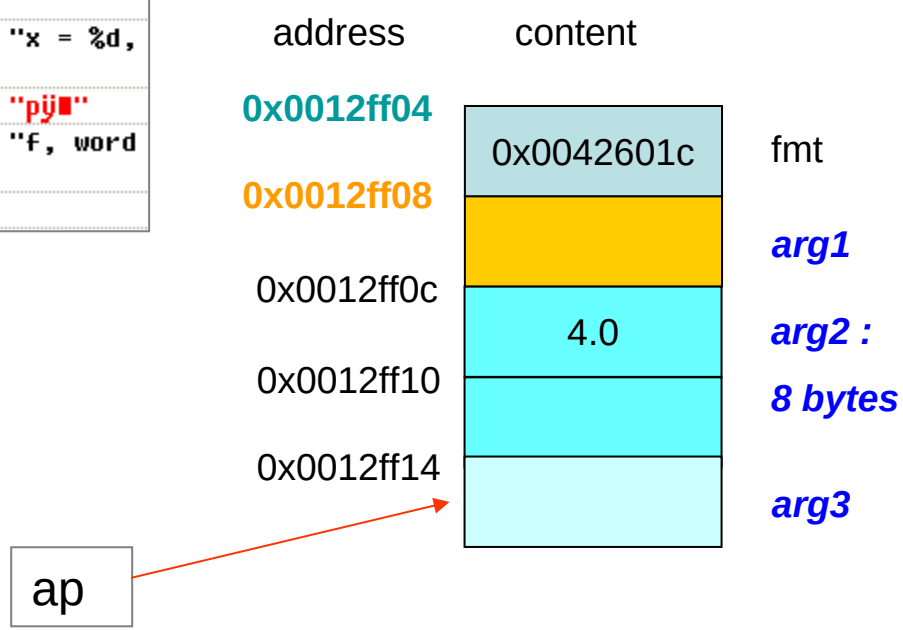




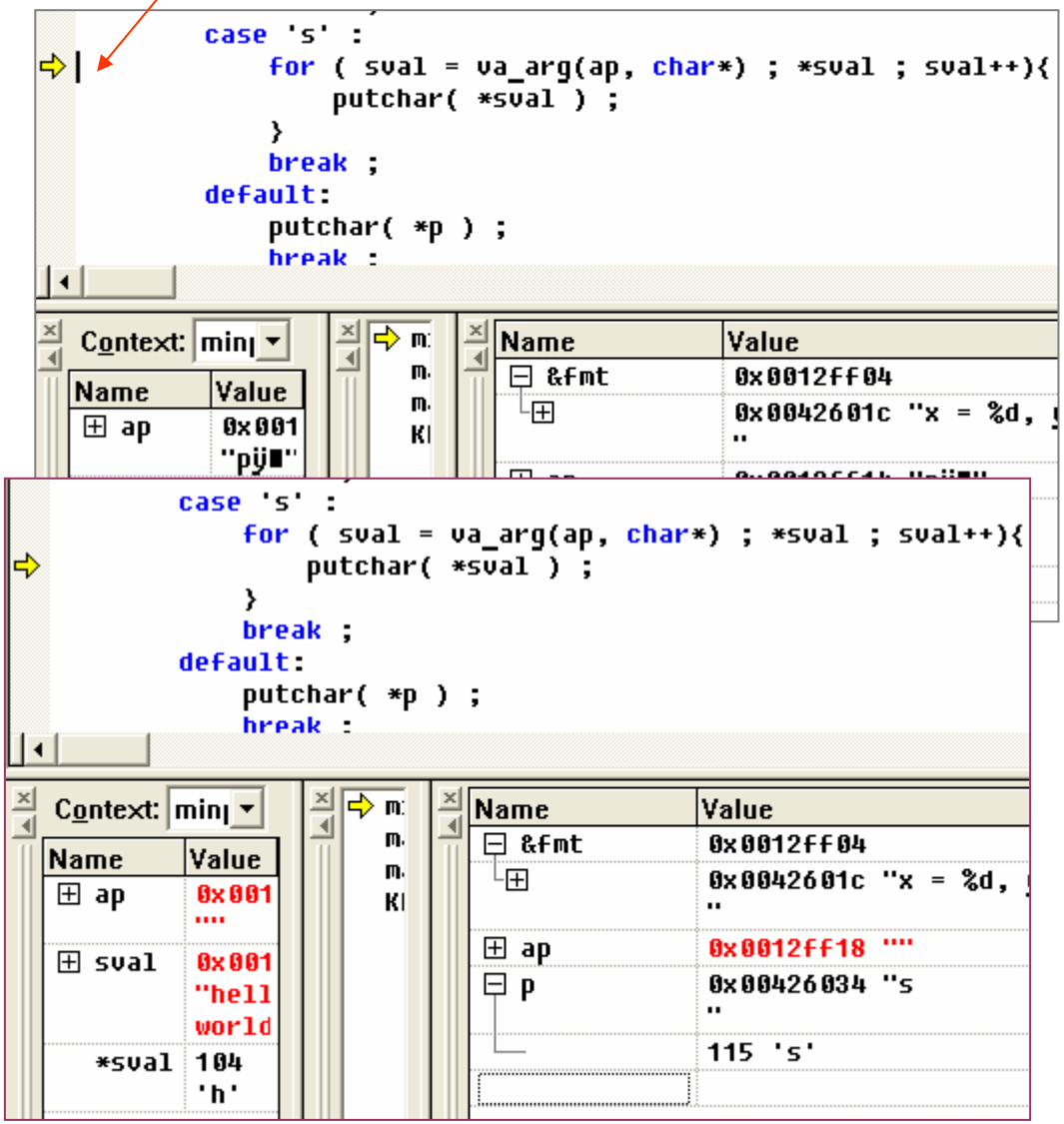
Debug → Go 至下一個中斷點

1. 以 double 型態取出 ***ap*** 所指的內容, 並存入 ***dval*** 中

2. 移動指標 ap 一個 double 的長度, 即 ap 指向下一個 argument (***arg3***)

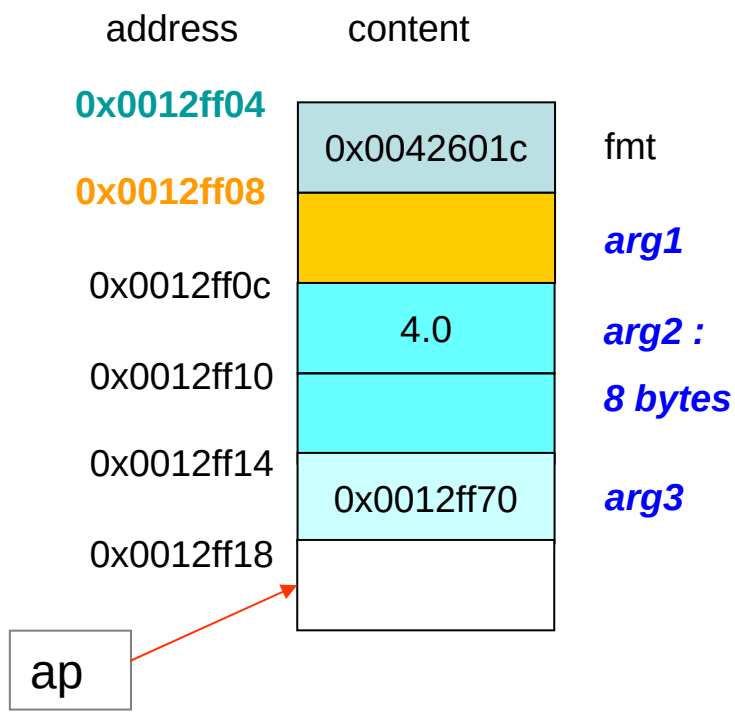


按 F10 執行　sval = va_arg(ap, char*)



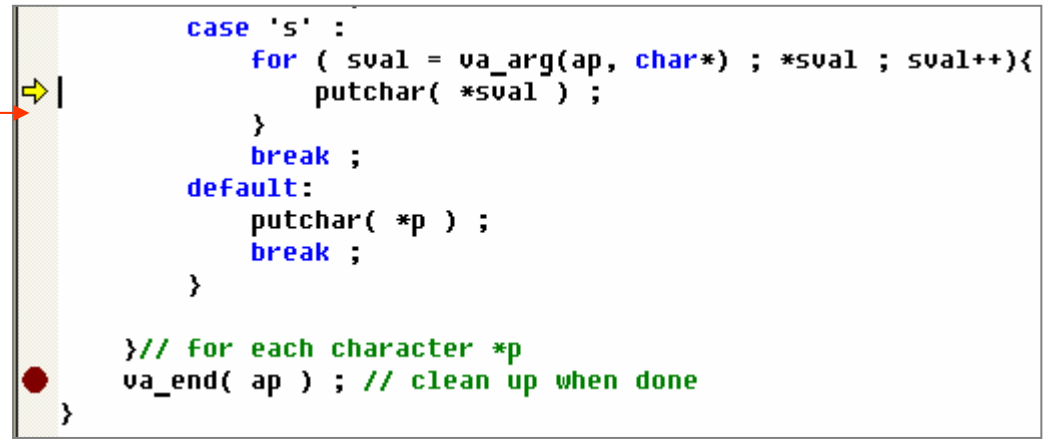
1. 以 char* 型態取出 ***ap*** 所指的內容, 並存入 ***sval*** 中

2. 移動指標 ap 一個 char* 的長度, 即 ap 指向下一個 argument (***arg4***)



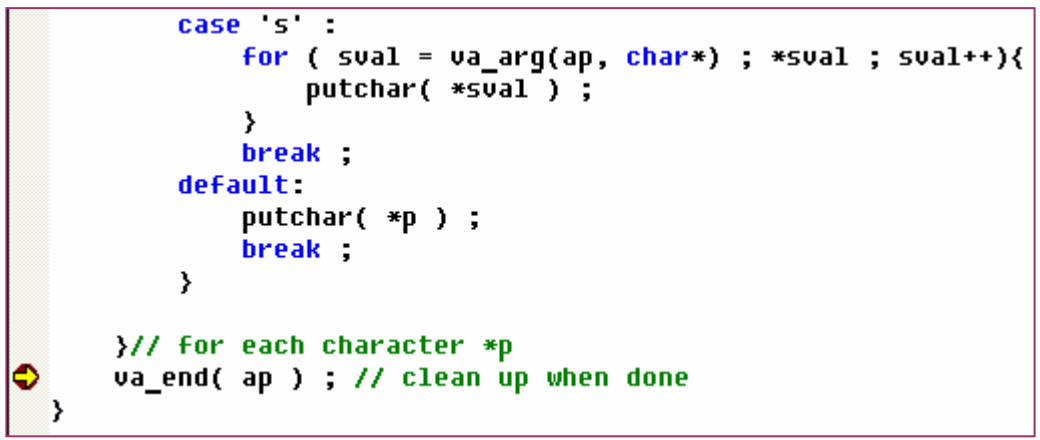
2. Debug → Go 至下一個中斷點

1. 按 F9 產生一個中斷點



```
        case 's' :
            for ( sval = va_arg(ap, char*) ; *sval ; sval++){
                putchar( *sval ) ;
            }
            break ;
        default:
            putchar( *p ) ;
            break ;
        }

    }// for each character *p
    va_end( ap ) ; // clean up when done
}
```

3.按 F10 執行 va_end



```
        case 's' :
            for ( sval = va_arg(ap, char*) ; *sval ; sval++){
                putchar( *sval ) ;
            }
            break ;
        default:
            putchar( *p ) ;
            break ;
        }

    }// for each character *p
    va_end( ap ) ; // clean up when done
}
```

Macro va_end

```
        case 's' :
            for ( sval = va_arg(ap, char*) ; *sval ; sval++){
                putchar( *sval ) ;
            }
            break ;
        default:
            putchar( *p ) ;
            break ;
        }

    }// for each character *p
    va_end( ap ) ; // clean up when done
}
```

Context: min

Name	Value
ap	0x00[illegible] ""

Name	Value
&fmt	0x0012ff04
ap	0x00000000 ""
	CXX0030: Error: expre
p	0x00426036 ""
sval	0x0012ff7c ""

address	content	
0x0012ff04	0x0042601c	fmt
0x0012ff08		*arg1*
0x0012ff0c	4.0	*arg2 :*
0x0012ff10		*8 bytes*
0x0012ff14	0x0012ff70	*arg3*
0x0012ff18		

Macro ***va_end*** resets pointer ***ap***

ap

0x00000000

Drawback of variable-length

- How to make sure consistence between *fmt* and number of parameters, arg1, arg2, ….
- *IMSL (線代及統計函式庫)* use \0 as final terminator, like string.
- Usual error in *printf* is mismatch of number of parameters. See homework

twins of printf – sprintf page 245

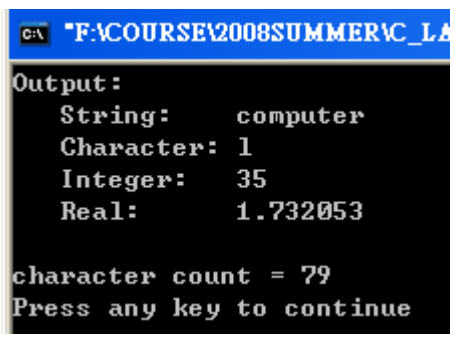
int ***sprintf*** (char* ***s***, const char ****fmt***, ***...***)

sprintf is the same as **printf** except that the ouput is written into the string **s**, terminated with '\0'.

String *s* must be big enough to hold the result.

Return value: the number of bytes stored in string **s**, not counting the terminating null character.

```
#include <stdio.h>

int main( void )
{
   char  buffer[200], s[] = "computer", c = '1';
   int   i = 35, j;
   float fp = 1.7320534f;

   // Format and print various data:
   j  = sprintf( buffer,     "   String:    %s\n", s ); // C4996
   j += sprintf( buffer + j, "   Character: %c\n", c ); // C4996
   j += sprintf( buffer + j, "   Integer:   %d\n", i ); // C4996
   j += sprintf( buffer + j, "   Real:      %f\n", fp );// C4996
   // Note: sprintf is deprecated; consider using sprintf_s instead

   printf( "Output:\n%s\ncharacter count = %d\n", buffer, j );

   return 0;
}
```



OutLine

- Format of printf
- Variable-length argument lists
- Formatted input – scanf
- File access
- Command execution

Formatted input – scanf

int ***scanf*** (char ****fmt***, ***...***)

int ***fscanf*** (FILE* stream, char ****fmt***, ***...***)

scanf (***fmt***, ***...***) = ***fscanf*** (***stdin***, ***fmt***, ***...***)

scanf reads characters from standard input, interprets them according to the specification in ***fmt***, and stores the results through the remaining arguments.

It returns number of input items converted and assigned.

Each of arguments must be a **pointer**, indicating where the corresponding converted input should be stored.

Question: why must arguments be pointers?

Format specification (from MSDN library)

```
%[*] [width] [{h | l | ll | I64 | L}]type
```

- ***width***
  The *width* field is a positive decimal integer controlling the maximum number of characters to be read for that field. No more than *width* characters are converted and stored at the corresponding argument.

- ***Size***
  The optional prefixes ***h***, ***l***, ***ll***, ***I64***, and ***L*** indicate the size of the argument.

- ***type***
  The type character determines whether the associated argument is interpreted as a character, string, or number.

Format specification: *type* *from MSDN*

Type Characters for scanf functions

Character	Type of input expected	Type of argument
c	Character. When used with **scanf** functions, specifies single-byte character; when used with **wscanf** functions, specifies wide character. White-space characters that are ordinarily skipped are read when **c** is specified. To read next non–white-space single-byte character, use **%1s**; to read next non–white-space wide character, use **%1ws**.	Pointer to **char** when used with **scanf** functions, pointer to **wchar_t** when used with **wscanf** functions.
d	Decimal integer.	Pointer to **int**.
i	Decimal, hexadecimal, or octal integer.	Pointer to **int**.
o	Octal integer.	Pointer to **int**.
u	Unsigned decimal integer.	Pointer to **unsigned int**.
x	Hexadecimal integer.	Pointer to **int**.
e, **E**, **f**, **g**, **G**	Floating-point value consisting of optional sign (+ or –), series of one or more decimal digits containing decimal point, and optional exponent ("e" or "E") followed by an optionally signed integer value.	Pointer to **float**.
s	String, up to first white-space character (space, tab or newline). To read strings not delimited by space characters, use set of square brackets (**[ ]**), as discussed in scanf Width Specification.	When used with **scanf** functions, signifies single-byte character array; when used with **wscanf** functions, signifies wide-character array. In either case, character array must be large enough for input field plus terminating null character, which is automatically appended.

Format specification: *size*

Size Prefixes for scanf and wscanf Format-Type Specifiers

To specify	Use prefix	With type specifier
double	**l**	**e**, **E**, **f**, **g**, or **G**
long double (same as double)	**L**	**e**, **E**, **f**, **g**, or **G**
long int	**l**	**d**, **i**, **o**, **x**, or **X**
long unsigned int	**l**	**u**
long long	**ll**	**d**, **i**, **o**, **x**, or **X**
short int	**h**	**d**, **i**, **o**, **x**, or **X**
short unsigned int	**h**	**u**
__int64	**I64**	**d**, **i**, **o**, **u**, **x**, or **X**
Single-byte character with **scanf**	**h**	**c** or **C**

Rudimentary calculator in page 158

% l f

type = *f* (floating-point) size = l (double 因爲 type = *f*)

```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    double  sum, v ;

    sum = 0.0 ;
    while( 1 == scanf("%lf", &v) ){
        printf("\t read v = %.2f, sum = ",v ) ;
        printf("\t%.2f\n", sum += v ) ;
    }
    return 0 ;
}
```

```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    double  sum, v ;

    sum = 0.0 ;
    while( 1 == scanf("%2lf", &v) ){
        printf("\t read v = %.2f, sum = ",v ) ;
        printf("\t%.2f\n", sum += v ) ;
    }
    return 0 ;
}
```

```
"F:\COURSE\2008SUMMER\C_LANG\EXAMPLE\CHAP7\calcu
10
        read v = 10.00, sum =  10.00
100
        read v = 100.00, sum =         110.00
1 1 1
        read v = 1.00, sum =   111.00
        read v = 1.00, sum =   112.00
        read v = 1.00, sum =   113.00
```

Question: what's the result?

OutLine

- Format of printf
- Variable-length argument lists
- Formatted input – scanf
- File access
- Command execution

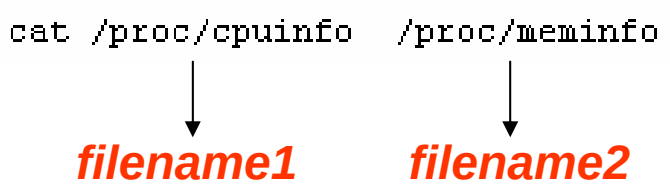
Example: cat (串連檔案)

```
[imsl@linux imsl]$ man cat

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION] [FILE]...

DESCRIPTION
       Concatenate FILE(s), or standard input, to standard output.
```

```
[imsl@linux imsl]$ cat /proc/cpuinfo  /proc/meminfo
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 3
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
bogomips        : 6016.20

        total:    used:    free:  shared: buffers:  cached:
Mem:  2113986560 470085632 1643900928        0 110436352 151838720
Swap: 2097434624        0 2097434624
MemTotal:      2064440 kB
MemFree:       1605372 kB
MemShared:           0 kB
Buffers:        107848 kB
Cached:         148280 kB
```

串連檔案 /proc/cpuinfo 和檔案 /proc/meminfo

Framework of cat



psudocode of *cat*

for each file name ***filename***

1. **open *filename***
2. **read** every character from filename and **output** to screen
3. **close *filename***

endfor

1. ***fopen*** : open a file for read/write
2. ***getc, putc*** : get character, put character
3. ***fclose*** : closed a file which is opened

Source code of cat

main.cpp

```
#include <stdio.h>

void filecopy( FILE *ifp, FILE *ofp ) ;

int main( int argc, char* argv[] )
{
1   FILE *fp ;

    if ( 1 == argc ){ // no args: copy standard input
        filecopy( stdin, stdout) ;
    }else{
        while( --argc >0 ){

2       fp = fopen( *++argv, "r" ) ;

            if ( NULL == fp ){
                printf("cat: can't open %s\n", *argv);
            }else{

3           filecopy( fp, stdout ) ;

4           fclose( fp ) ;

            }
        }// for each argument
    }
    return 0 ;
}
```

filecopy.cpp

```
#include <stdio.h>

void filecopy( FILE *ifp, FILE *ofp )
{
    int c ;

    while ( EOF != (c = getc(ifp)) ){
        putc(c, ofp ) ;
    }
}
```

or

```
#include <stdio.h>

void filecopy( FILE *ifp, FILE *ofp )
{
    int c ;

    while ( EOF != (c = fgetc(ifp)) ){
        fputc(c, ofp ) ;
    }
}
```

Type *FILE* in stdio.h

Visual studio

```
#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
```

Linux radhat9

```
/* The opaque type of streams.  This is the definition used elsewhere.  */
typedef struct _IO_FILE FILE;
```

```
/*
 * Number of entries in _iob[] (declared below). Note that _NSTREAM_ must be
 * greater than or equal to _IOB_ENTRIES.
 */
#define _IOB_ENTRIES 20
/* Declare _iob[] array */

#ifndef _STDIO_DEFINED
_CRTIMP extern FILE _iob[];
#endif  /* _STDIO_DEFINED */
```

```
#define stdin  (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])
```

```
/* Standard streams.  */
extern struct _IO_FILE *stdin;
extern struct _IO_FILE *stdout;
extern struct _IO_FILE *stderr;
```

FILE* ***fopen*** (const char ****filename***, const char ****mode***)

mode	description
"r"	Opens for reading. If the file does not exist or cannot be found, the ***fopen*** call fails.
"w"	Opens an empty file for writing. If the given file exists, its contents are destroyed.
"a"	Opens for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; creates the file first if it doesn't exist.
"r+"	Opens for both reading and writing. (The file must exist.)
"w+"	Opens an empty file for both reading and writing. If the given file exists, its contents are destroyed.
"a+"	Opens for reading and appending; the appending operation includes the removal of the EOF marker before new data is written to the file and the EOF marker is restored after writing is complete; creates the file first if it doesn't exist.

- 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.

- int ***getc*** (FILE* *stream*)
  getc is equivalent to *fgetc* except that if it is a macro, it may evaluate *stream* more than once

- int ***fputc*** (int c, FILE* *stream*)
  fputc writes the character c (converted to an *unsigned char*) on *stream.* It returns the character written, or EOF for an error.

- Int ***putc*** (int c, FILE* *stream*)
  putc is equivalent to *fputc* except that if it is a macro, it may evaluate *stream* more than once

- int ***fclose*** (FILE* **stream**)
  fclose flushes any unwritten data for *stream*, discards any unread buffered input, frees any automatically allocated buffer, then close the *stream*. It returns EOF if any errors occurred, and zero otherwise.

Summary of standard IO

- When a C program started, the OS (operating system 作業系統) environment is responsible for opening three files (standard input, standard output and standard error) and providing file pointers for them.

- File pointers are called *stdin*, *stdout*, *stderr* declared in stdio.h
- *stdin* is connected to keyboard and *stdout* is connected to the screen, but *stdin* and *stdout* may be re-directed to files or pipes.

- *getchar* and *putchar* can be defined in terms of *getc*, *putc*, *stdin* and *stdout*
  #define *getchar*( ) *getc*(*stdin*)
  #define *putchar*(c) *putc*((c), *stdout*)

- *scanf* and *printf* can be defined in terms of *fscanf* and *fprintf*
  int *fscanf*(FILE *fp, char ***fmt**, **...**)
  int *fprintf*(FILE *fp, char ***fmt**, **...**)

OutLine

- Format of printf
- Variable-length argument lists
- Formatted input – scanf
- File access
- Command execution

Command execution

int ***system*** (const char ***s***)

system passes the string *s* to the environment for execution. If *s* is NULL, return non-zero if there is a command processor. If *s* is not NULL, the return value is implementation-dependent.

```
#include <stdlib.h>

int main( void )
{

#ifdef _WIN32
    system( "dir" );
#else
    system( "ls -al" ) ;
#endif

   return 0 ;
}
```

```
[imsl@linux system]$ ls
Debug  main.cpp  system.dsp  system.dsw  system.ncb  system.opt  system.plg
[imsl@linux system]$ icpc main.cpp
[imsl@linux system]$ ls
a.out  main.cpp    system.dsw  system.opt
Debug  system.dsp  system.ncb  system.plg
[imsl@linux system]$ ./a.out
total 108
drwxr-xr-x    3 imsl     imsl         4096 Jul  8  2008 .
drwxr-xr-x    6 imsl     imsl         4096 Jul  8  2008 ..
-rwxrwxr-x    1 imsl     imsl        25280 Jul  8 10:50 a.out
drwxr-xr-x    2 imsl     imsl         4096 Jul  8  2008 Debug
-rw-r--r--    1 imsl     imsl          142 Jul  8  2008 main.cpp
-rw-r--r--    1 imsl     imsl         4660 Jul  8  2008 system.dsp
-rw-r--r--    1 imsl     imsl          535 Jul  8  2008 system.dsw
-rw-rw-r--    1 imsl     imsl            0 Jul  8  2008 system.ncb
-rw-r--r--    1 imsl     imsl        48640 Jul  8  2008 system.opt
-rw-r--r--    1 imsl     imsl         1289 Jul  8  2008 system.plg
[imsl@linux system]$
```

```
"F:\course\2008summer\c_lang\example\chap7\system\Debug\system.exe"
 磁碟區 F 中的磁碟沒有標籤。
 磁碟區序號:  C065-A3C9

 F:\course\2008summer\c_lang\example\chap7\system 的目錄

2008/07/08  上午 11:02    <DIR>          .
2008/07/08  上午 11:02    <DIR>          ..
2008/07/08  上午 11:02    <DIR>          Debug
2008/07/08  上午 11:02               146 main.cpp
2008/07/08  上午 10:56             4,660 system.dsp
2008/07/08  上午 10:55               535 system.dsw
2008/07/08  上午 10:55            25,600 system.ncb
2008/07/08  上午 11:02            48,640 system.opt
2008/07/08  上午 11:02             1,289 system.plg
               6 個檔案          80,870 位元組
               3 個目錄  11,673,010,176 位元組可用
Press any key to continue_
```