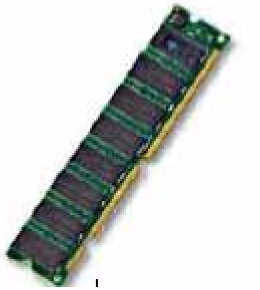
Chapter 5
pointers and arrays

Speaker: Lung-Sheng Chien

OutLine

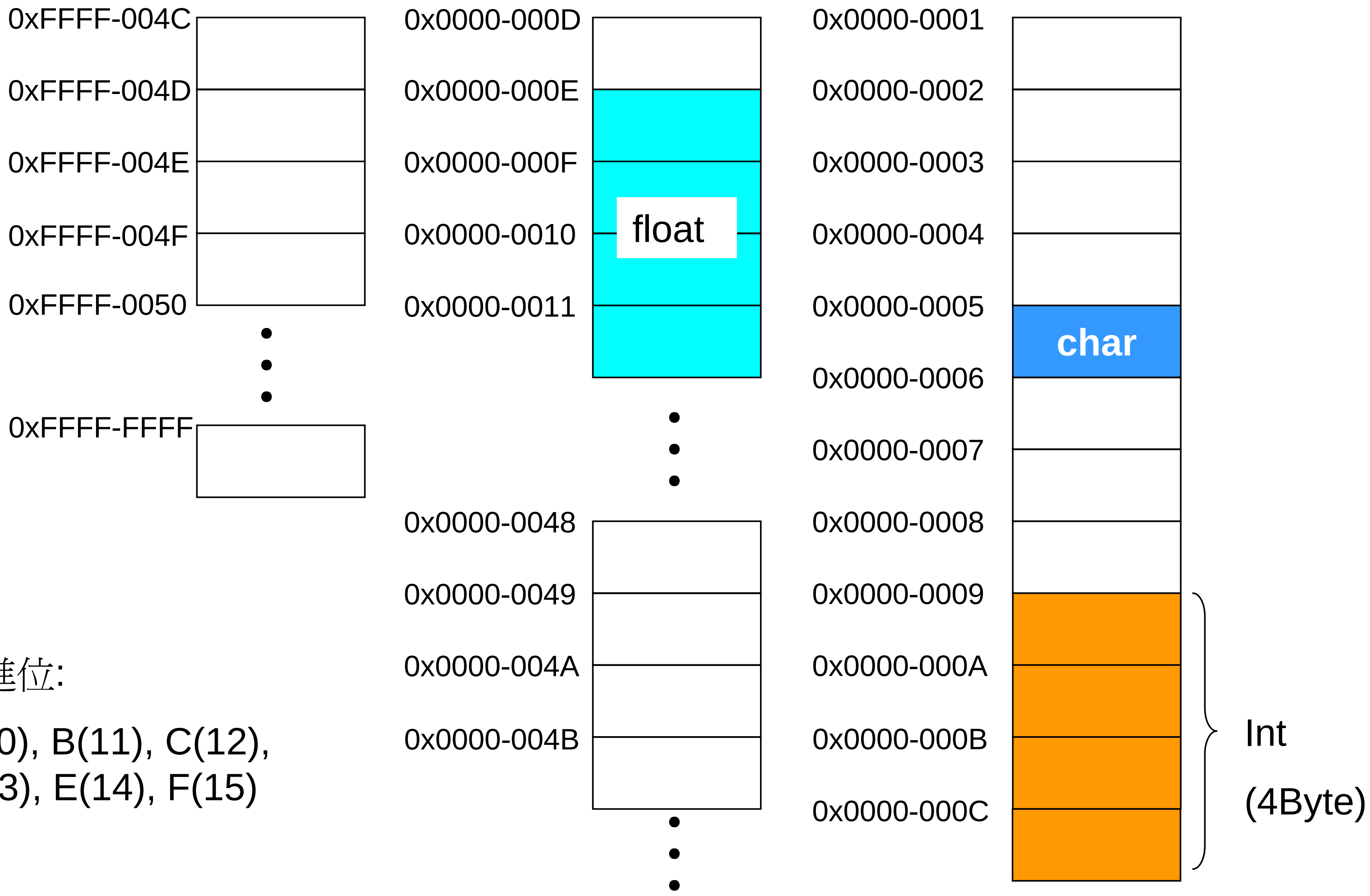
- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

Physical memory



- Basic unit of memory is byte (8 bits)
- Each memory unit (byte) needs an unique address to identify it.
- For 32-bit system, we can address memory from 0 (0x00000000) to $2^{32}-1 \sim 4\times10^{9}$ (0xFFFFFFFF), total memory is up to 4GB
- For 64-bit system, we can address memory from 0 (0x00000000-00000000) to $2^{64}-1 \sim 16\times10^{18}$ (0xFFFFFFFF-FFFFFFFF), total memory is ∞
- unit: kB = 1000 bytes, 1MB = 1000 kB, 1GB = 1000 MB

1-dimensional memory block in 32-bit system

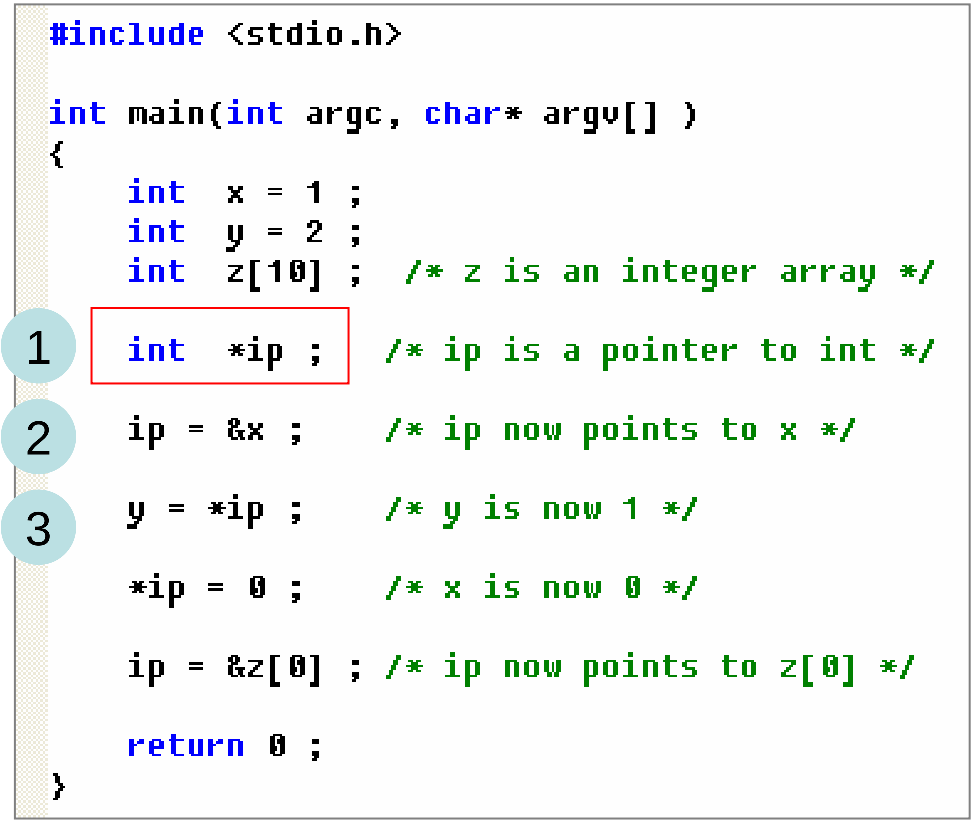


16進位:

A(10), B(11), C(12),
D(13), E(14), F(15)

What is pointer

A pointer is a variable that contains the address of a variable



```
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  x = 1 ;
    int  y = 2 ;
    int  z[10] ;  /* z is an integer array */

    int  *ip ;   /* ip is a pointer to int */

    ip = &x ;    /* ip now points to x */

    y = *ip ;    /* y is now 1 */

    *ip = 0 ;    /* x is now 0 */

    ip = &z[0] ; /* ip now points to z[0] */

    return 0 ;
}
```

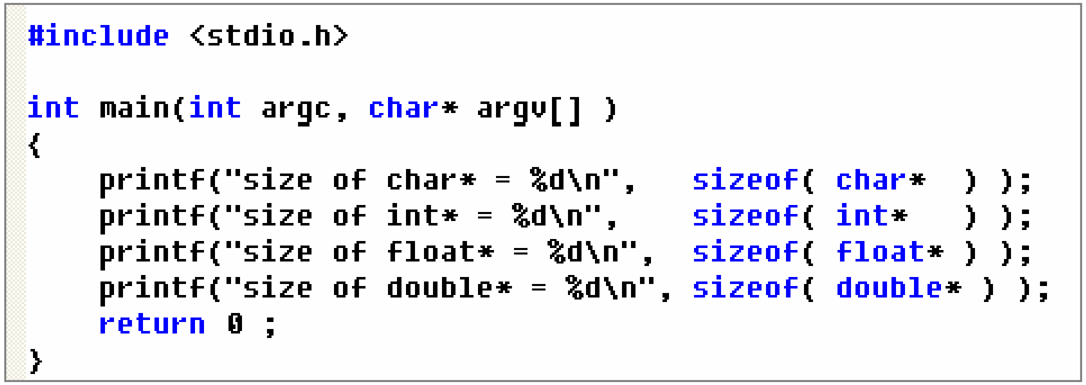
1. 宣告 ip 是個整數指標

2. **&** is called reference operator, 提取 x 的 address位址給指標 ip

3. ***** is called dereference operator, 將指標 ip 所存的值當作記憶體位址, 則 *ip 即是變數 x

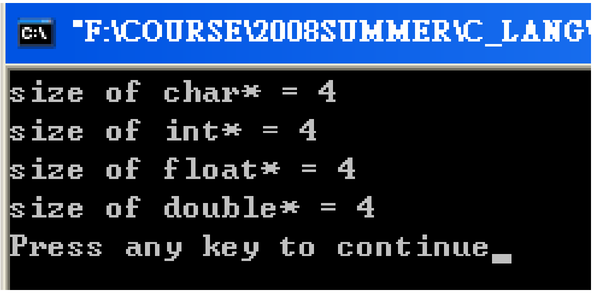
Question: Since pointer is also a variable, then address = ?, size = ?

Size of pointer type



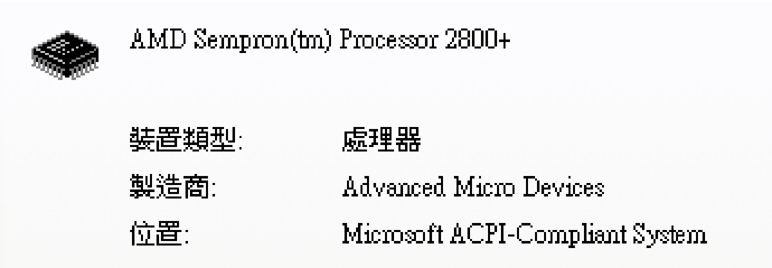
```c
#include <stdio.h>

int main(int argc, char* argv[] )
{
    printf("size of char* = %d\n",   sizeof( char*  ) );
    printf("size of int* = %d\n",    sizeof( int*   ) );
    printf("size of float* = %d\n",  sizeof( float* ) );
    printf("size of double* = %d\n", sizeof( double* ) );
    return 0 ;
}
```

result in windows



"F:\COURSE\2008SUMMER\C_LANG

```
size of char* = 4
size of int* = 4
size of float* = 4
size of double* = 4
Press any key to continue_
```



AMD Sempron(tm) Processor 2800+

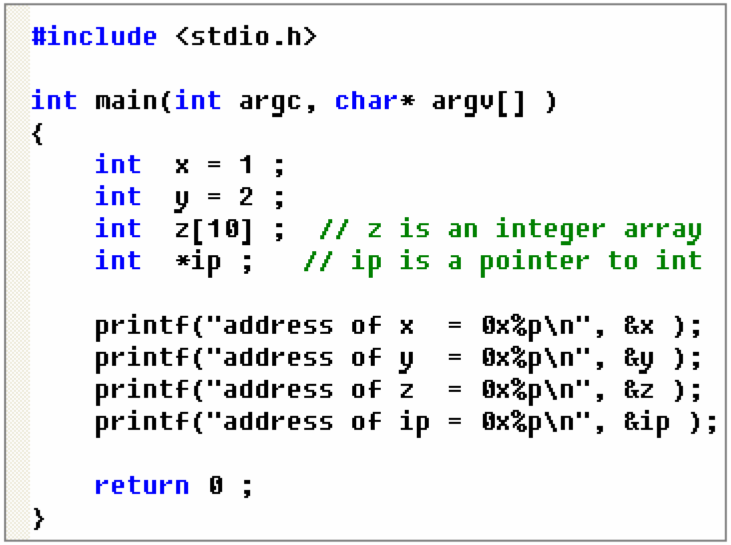
裝置類型:	處理器
製造商:	Advanced Micro Devices
位置:	Microsoft ACPI-Compliant System

Question: all four pointer types have size 4 bytes, why?

A pointer is a variable that contains the address of a variable

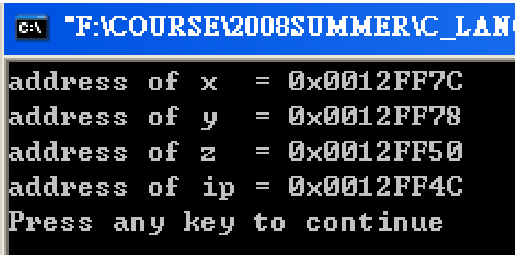
32-bit machine uses 32 bit to address a variable

Address of a variable



```
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  x = 1 ;
    int  y = 2 ;
    int  z[10] ;  // z is an integer array
    int  *ip ;   // ip is a pointer to int

    printf("address of x  = 0x%p\n", &x );
    printf("address of y  = 0x%p\n", &y );
    printf("address of z  = 0x%p\n", &z );
    printf("address of ip = 0x%p\n", &ip );

    return 0 ;
}
```

result in windows

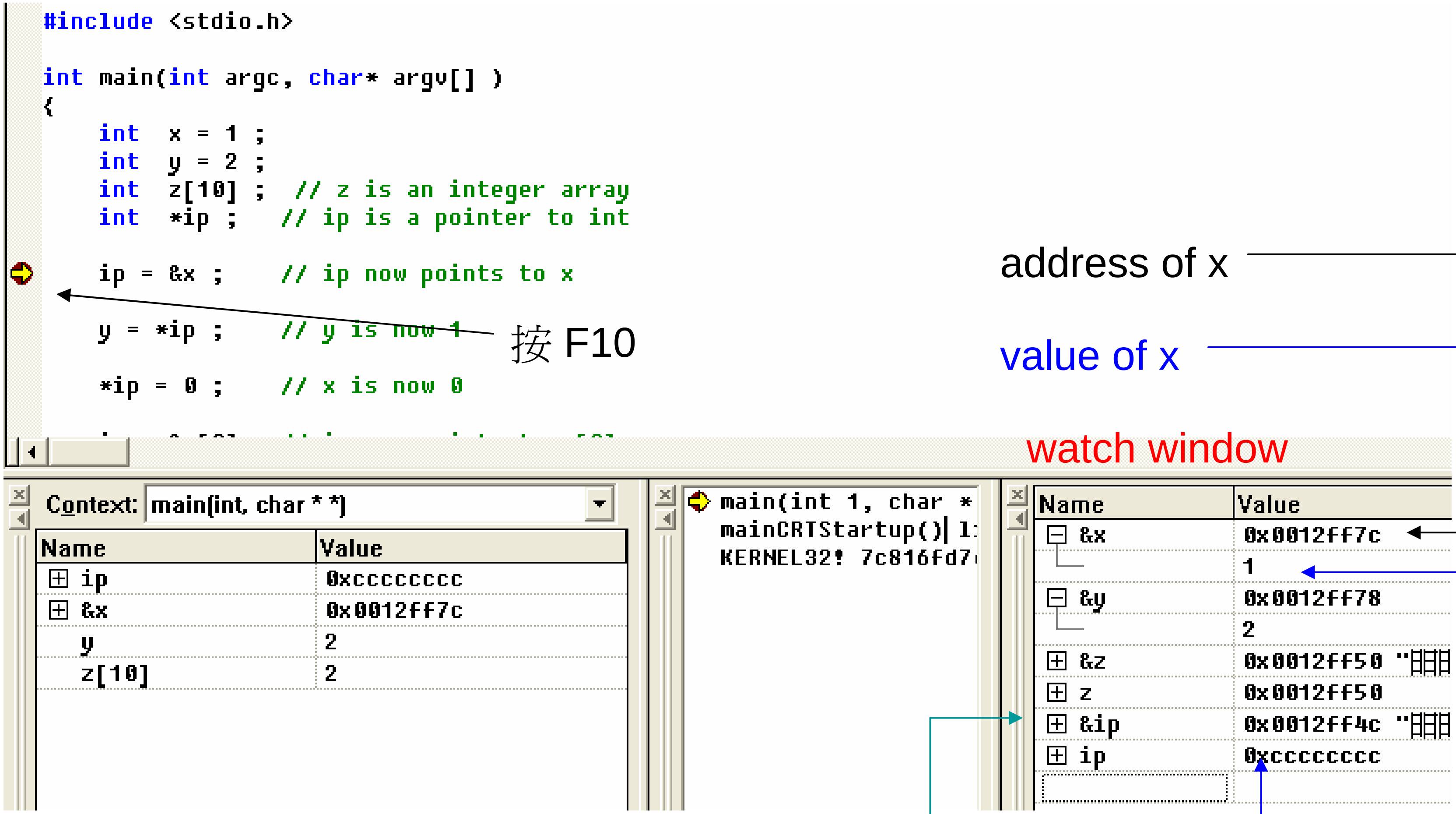


```
address of x  = 0x0012FF7C
address of y  = 0x0012FF78
address of z  = 0x0012FF50
address of ip = 0x0012FF4C
Press any key to continue
```

16進位: A(10), B(11), C(12), D(13),
E(14), F(15)

address	content	variable
0x0012FF4C	?	ip
0x0012FF50	?	z[0]
0x0012FF54	?	z[1]
0x0012FF58	?	z[2]
0x0012FF5C	?	z[3]
0x0012FF60	?	z[4]
0x0012FF64	?	z[5]
0x0012FF68	?	z[6]
0x0012FF6C	?	z[7]
0x0012FF70	?	z[8]
0x0012FF74	?	z[9]
0x0012FF78	2	y
0x0012FF7C	1	x

Use debugger to show change of memory [1]



```
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  x = 1 ;
    int  y = 2 ;
    int  z[10] ;  // z is an integer array
    int  *ip ;   // ip is a pointer to int

    ip = &x ;    // ip now points to x

    y = *ip ;    // y is now 1

    *ip = 0 ;    // x is now 0
```

按 F10

address of x

value of x

watch window

Context: main(int, char * *)

Name	Value
⊞ ip	0xcccccccc
⊞ &x	0x0012ff7c
y	2
z[10]	2

```
main(int 1, char *
mainCRTStartup() l
KERNEL32! 7c816fd7
```

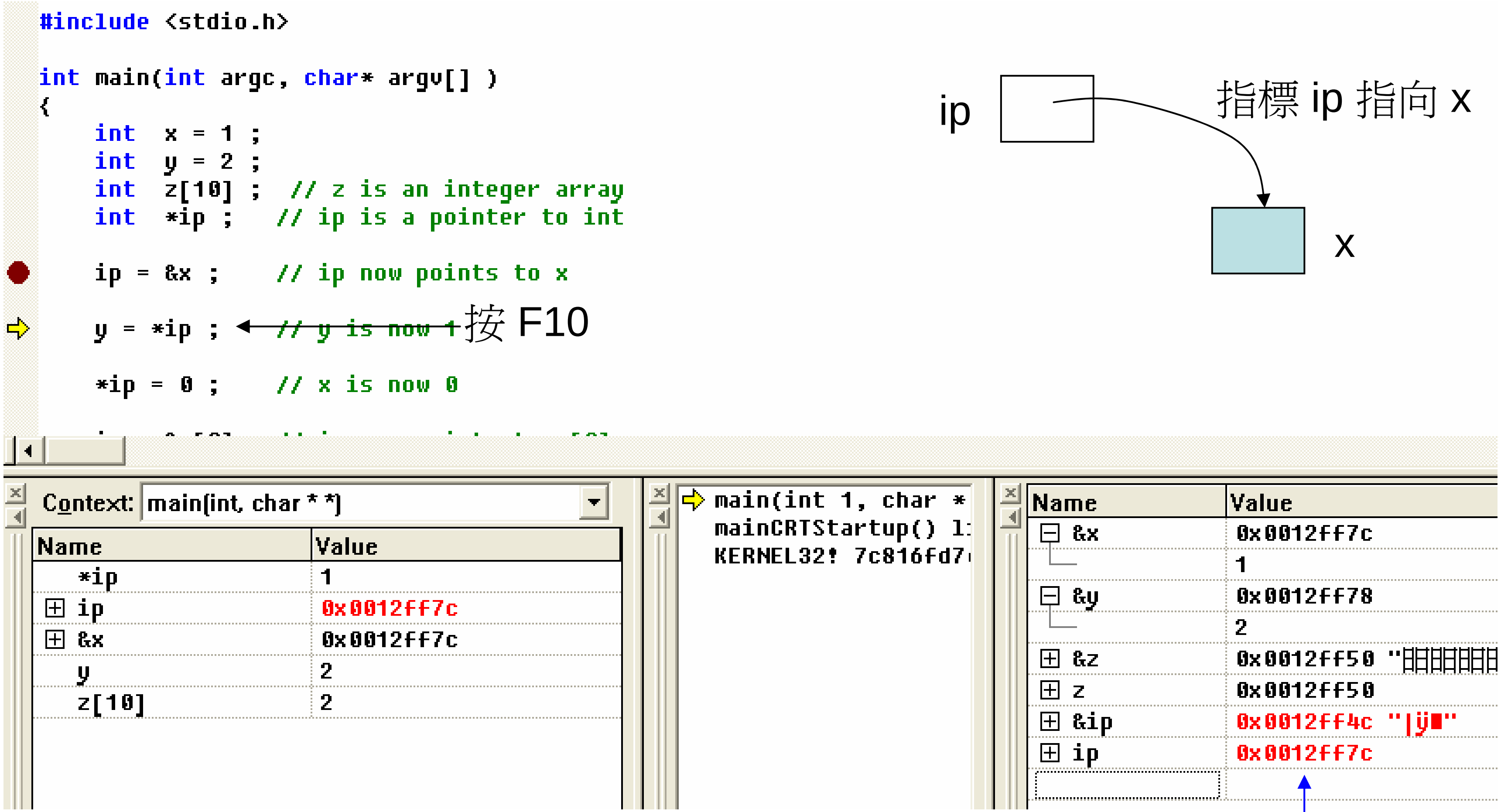
Name	Value
⊟ &x	0x0012ff7c
	1
⊟ &y	0x0012ff78
	2
⊞ &z	0x0012ff50 "
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "
⊞ ip	0xcccccccc

address of pointer ip

value of ip (address of some varable)

: 0xcccccccc means invalid

Use debugger to show change of memory   [2]



```
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  x = 1 ;
    int  y = 2 ;
    int  z[10] ;  // z is an integer array
    int  *ip ;   // ip is a pointer to int

    ip = &x ;    // ip now points to x

    y = *ip ;    // y is now 1

    *ip = 0 ;    // x is now 0
```

Context: main(int, char * *)

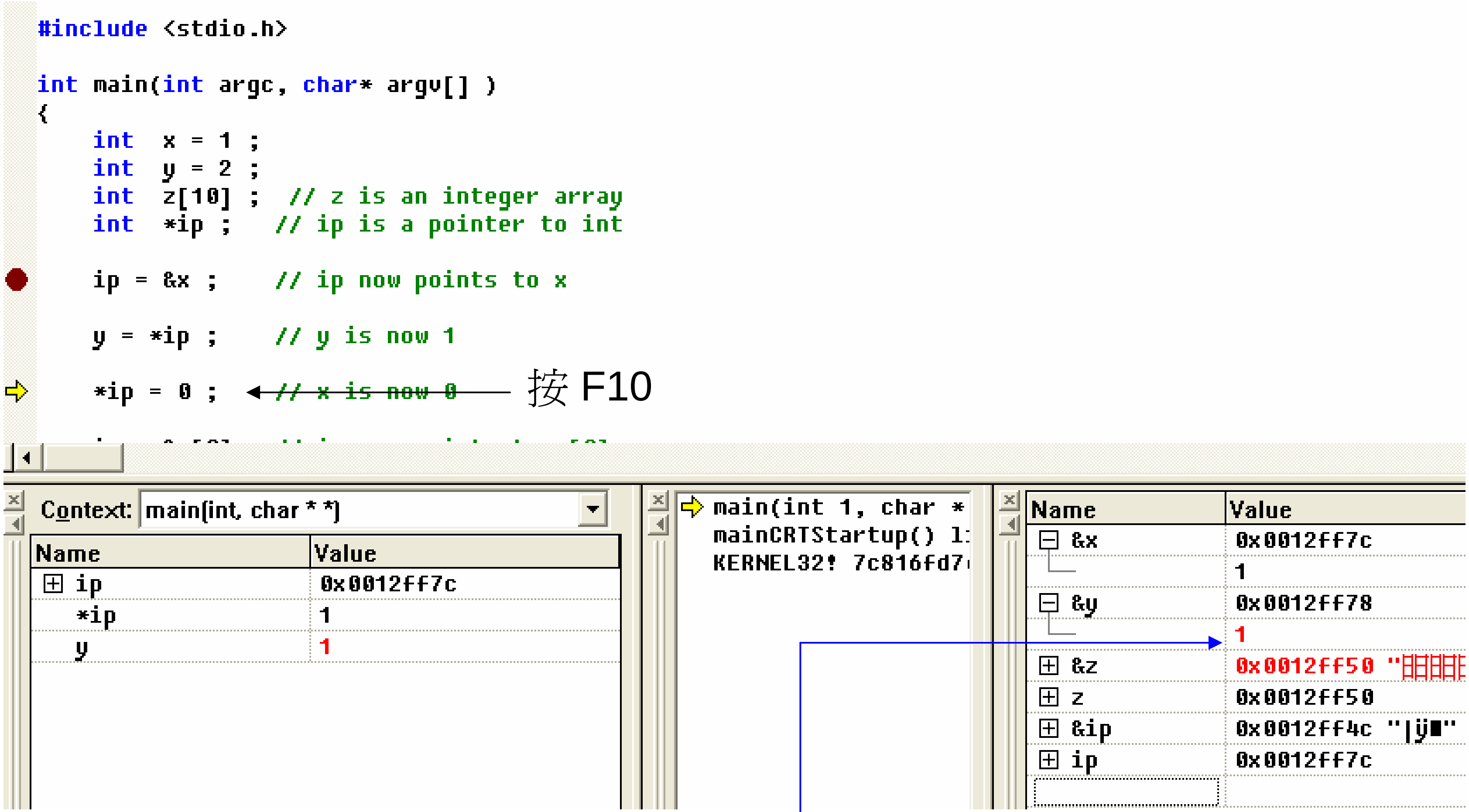
Name	Value
*ip	1
⊞ ip	0x0012ff7c
⊞ &x	0x0012ff7c
y	2
z[10]	2

```
main(int 1, char *
mainCRTStartup() 1
KERNEL32! 7c816fd7
```

Name	Value
⊟ &x	0x0012ff7c
	1
⊟ &y	0x0012ff78
	2
⊞ &z	0x0012ff50 "昍昍昍昍昍昍
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "|ÿ■"
⊞ ip	0x0012ff7c

value of ip = address of x

Use debugger to show change of memory [3]



```
#include <stdio.h>

int main(int argc, char* argv[] )
{
	int  x = 1 ;
	int  y = 2 ;
	int  z[10] ;  // z is an integer array
	int  *ip ;   // ip is a pointer to int

	ip = &x ;    // ip now points to x

	y = *ip ;    // y is now 1

	*ip = 0 ;  ~~// x is now 0~~
```

按 F10

Context: main(int, char * *)

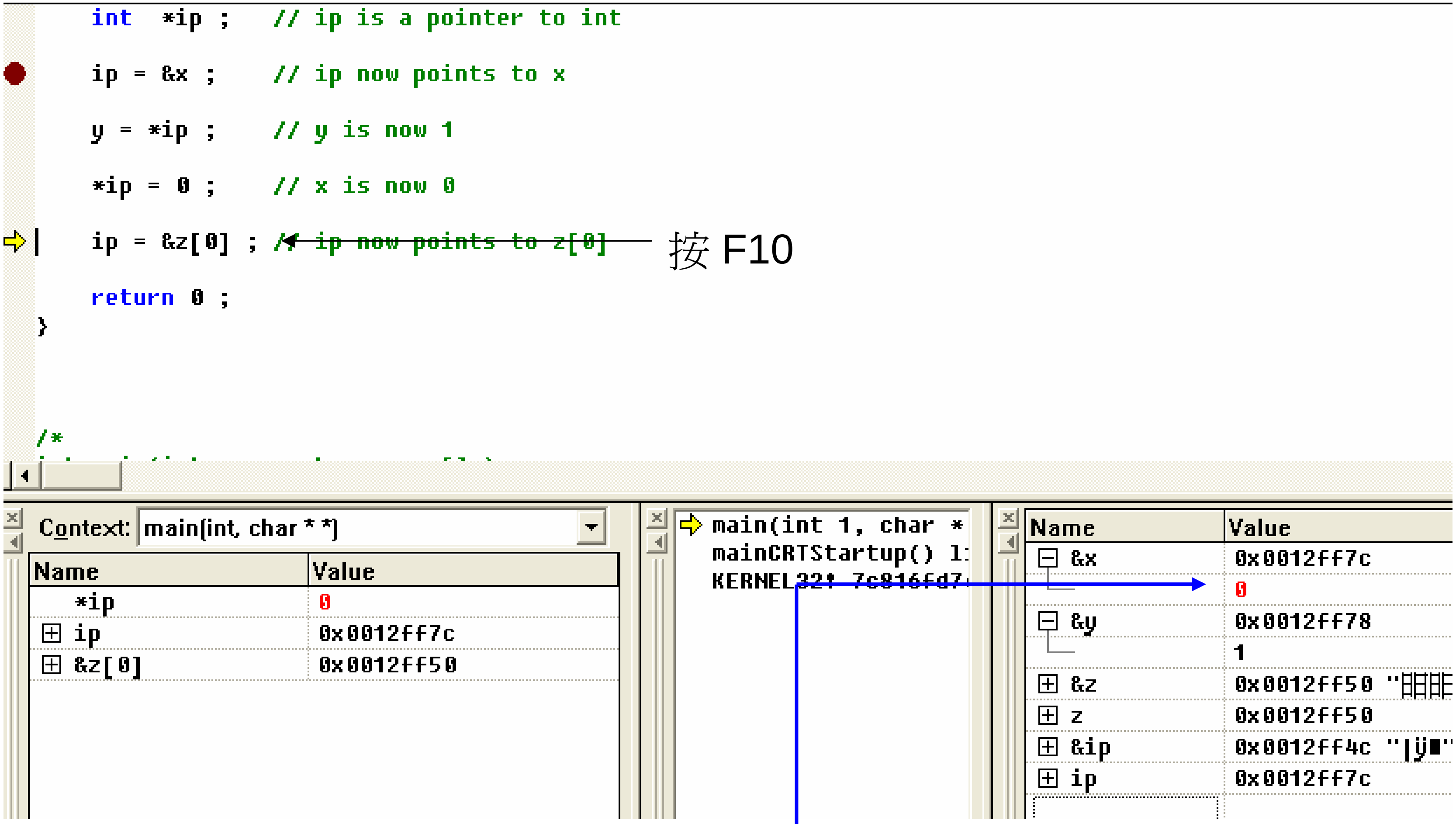
Name	Value
⊞ ip	0x0012ff7c
*ip	1
y	1

```
main(int 1, char *
mainCRTStartup() l
KERNEL32! 7c816fd7
```

Name	Value
⊟ &x	0x0012ff7c
	1
⊟ &y	0x0012ff78
	1
⊞ &z	0x0012ff50 "ĦĦĦĦ
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "|ÿ■"
⊞ ip	0x0012ff7c

因爲 *ip 即爲 x, 所以 y = *ip 和 y = x 等價, 即 y 被設爲 1

Use debugger to show change of memory [4]



```
    int  *ip ;  // ip is a pointer to int

    ip = &x ;   // ip now points to x

    y = *ip ;   // y is now 1

    *ip = 0 ;   // x is now 0

    ip = &z[0] ; // ip now points to z[0]

    return 0 ;
}


/*
```

按 F10

Context: main(int, char * *)

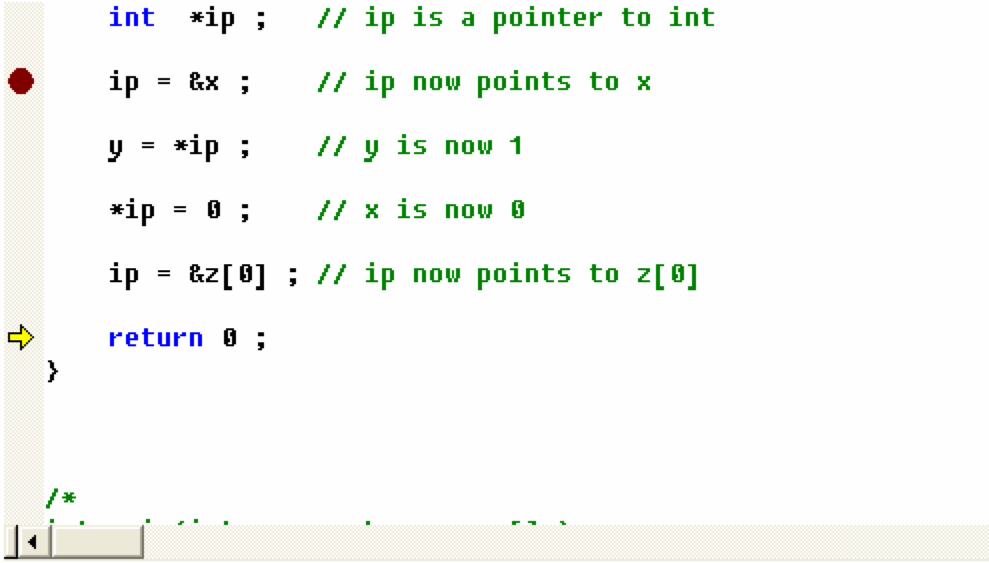
Name	Value
*ip	0
⊞ ip	0x0012ff7c
⊞ &z[0]	0x0012ff50

```
main(int 1, char *
mainCRTStartup() l:
KERNEL32! 7c816fd7
```

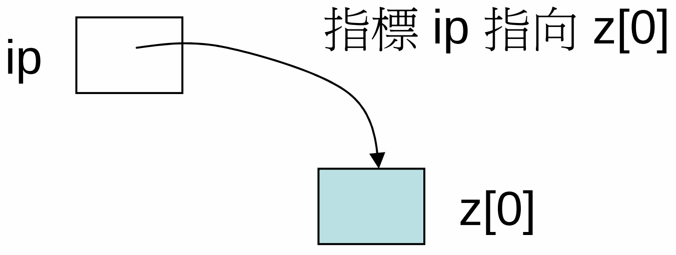
Name	Value
⊟ &x	0x0012ff7c
	0
⊟ &y	0x0012ff78
	1
⊞ &z	0x0012ff50 "昍昍
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "|ÿ■"
⊞ ip	0x0012ff7c

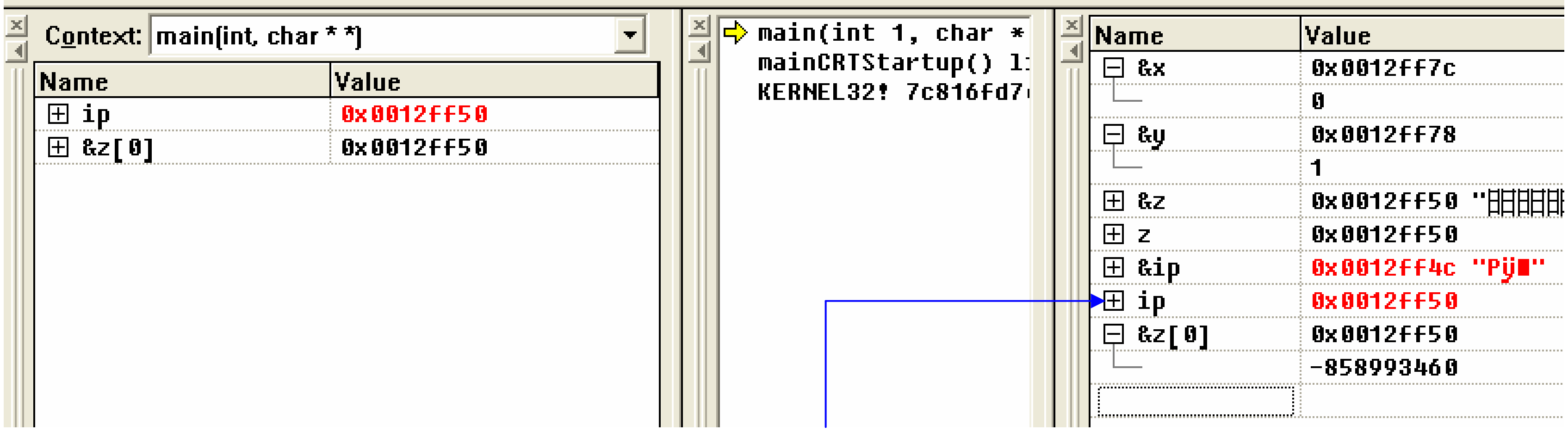
因爲 *ip 即爲 x, 所以 *ip = 0 和 x = 0 等價, 即 x 被設爲 0

Use debugger to show change of memory [5]



```
int  *ip ;   // ip is a pointer to int

ip = &x ;    // ip now points to x

y = *ip ;    // y is now 1

*ip = 0 ;    // x is now 0

ip = &z[0] ; // ip now points to z[0]

return 0 ;
}


/*
```





Context: main(int, char * *)

Name	Value
⊞ ip	0x0012ff50
⊞ &z[0]	0x0012ff50

```
main(int 1, char *
mainCRTStartup() l
KERNEL32! 7c816fd7
```

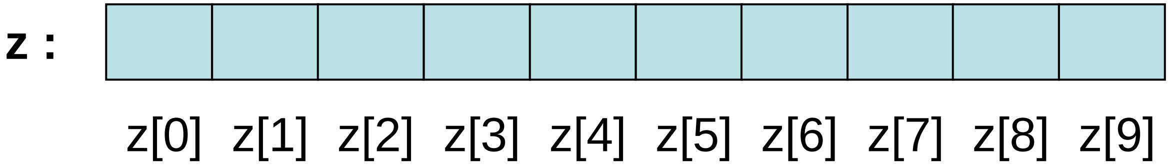
Name	Value
⊟ &x	0x0012ff7c
	0
⊟ &y	0x0012ff78
	1
⊞ &z	0x0012ff50 "□□□□□
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "Pÿ■"
⊞ ip	0x0012ff50
⊟ &z[0]	0x0012ff50
	-858993460

ip = &z[0] 提取 z[0] 的位址並存入指標 ip 內

OutLine

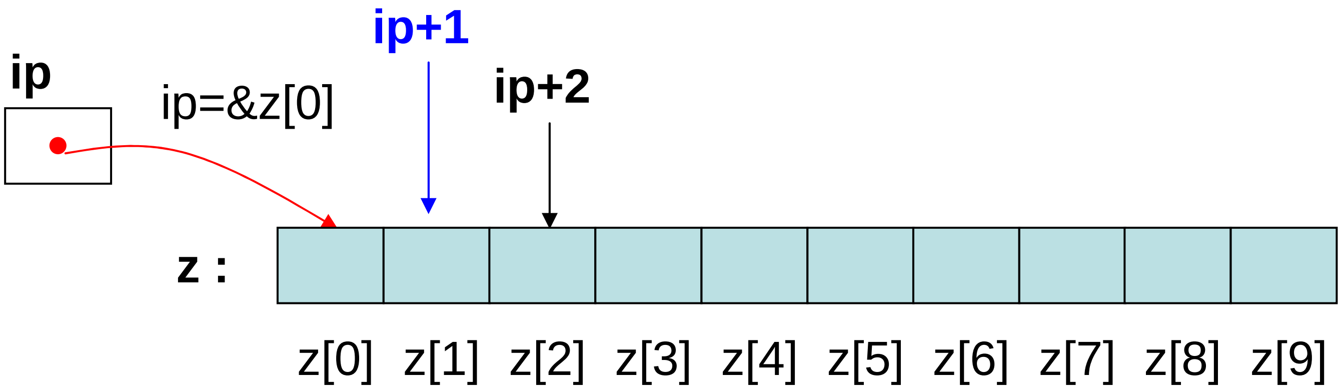
- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

Pointer and array [1]



z :

z[0] z[1] z[2] z[3] z[4] z[5] z[6] z[7] z[8] z[9]



ip

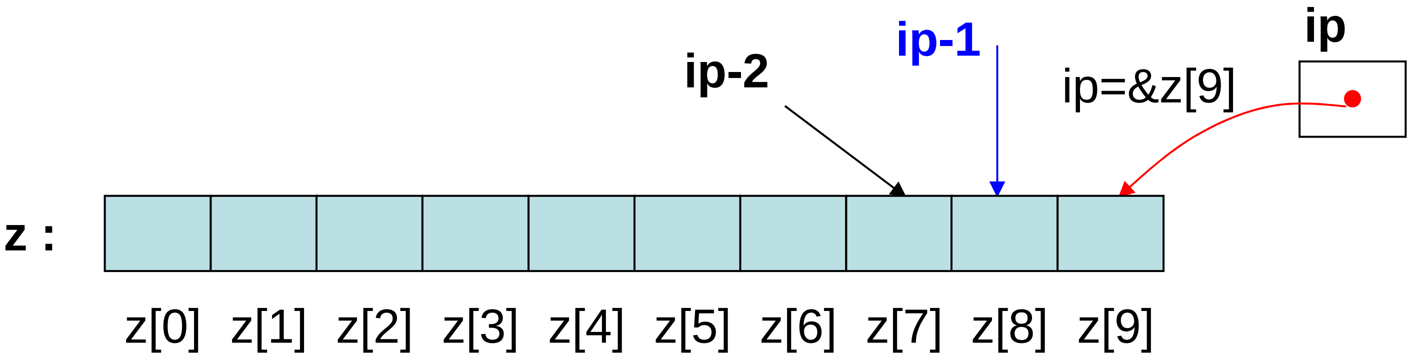
ip=&z[0]

ip+1

ip+2

z :

z[0] z[1] z[2] z[3] z[4] z[5] z[6] z[7] z[8] z[9]



ip-2

ip-1

ip=&z[9]

ip

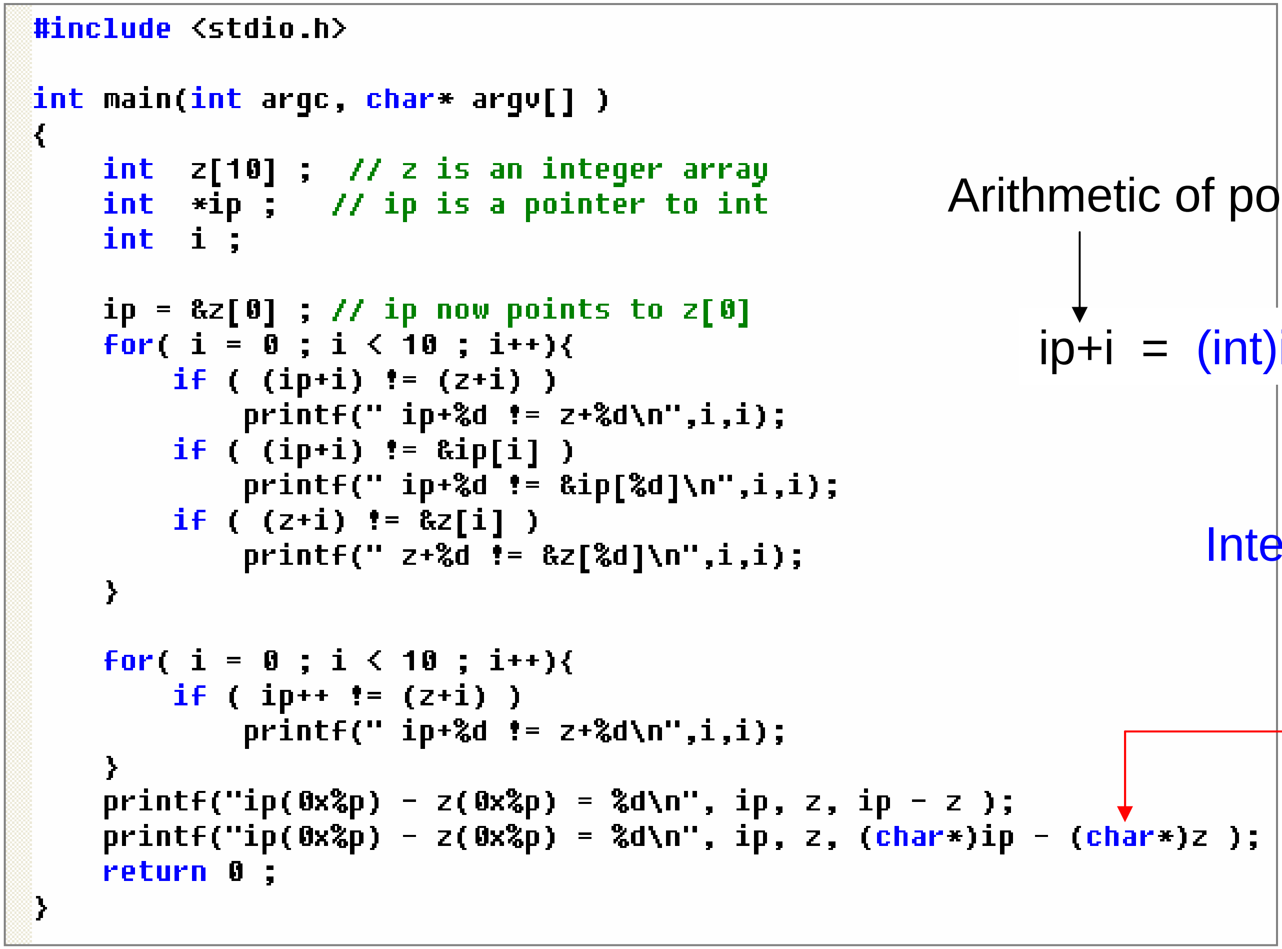
z :

z[0] z[1] z[2] z[3] z[4] z[5] z[6] z[7] z[8] z[9]

Pointer and array [2]

- ip := z is equivalent to ip := &z[0] since array name z is synonym(同義字) for first element of z, z[0]. In other words, z = &z[0]
- z[i] = *(z+i), this is default substitution in C-language. In other words, &z[i] = z+i
- ip+i points to i-th object beyond ip
- If ip:=z, then ip plays the same role as z, say z[i] = *(ip+i) = *(z+i) = ip[i] , or &z[i] = ip+I = z+I = &ip[i]
- ip++ is equivalent to ip = ip+1, (move pointer ip to next object). However z is name of array, it always points to first element z[0], so z++ is illegal. You may say pointer is a movable array name.
- ip+1 is an integer, but *(ip+1) is a reference to some memory location, l-value

Pointer and array [3]



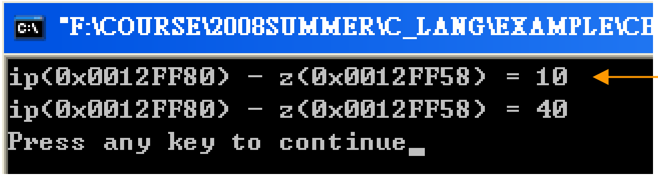
```
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  z[10] ;  // z is an integer array
    int  *ip ;   // ip is a pointer to int
    int  i ;

    ip = &z[0] ; // ip now points to z[0]
    for( i = 0 ; i < 10 ; i++){
        if ( (ip+i) != (z+i) )
            printf(" ip+%d != z+%d\n",i,i);
        if ( (ip+i) != &ip[i] )
            printf(" ip+%d != &ip[%d]\n",i,i);
        if ( (z+i) != &z[i] )
            printf(" z+%d != &z[%d]\n",i,i);
    }

    for( i = 0 ; i < 10 ; i++){
        if ( ip++ != (z+i) )
            printf(" ip+%d != z+%d\n",i,i);
    }
    printf("ip(0x%p) - z(0x%p) = %d\n", ip, z, ip - z );
    printf("ip(0x%p) - z(0x%p) = %d\n", ip, z, (char*)ip - (char*)z );
    return 0 ;
}
```

Arithmetic of pointer

ip+i = (int)ip + sizeof(int)*i

Integral arithmetic

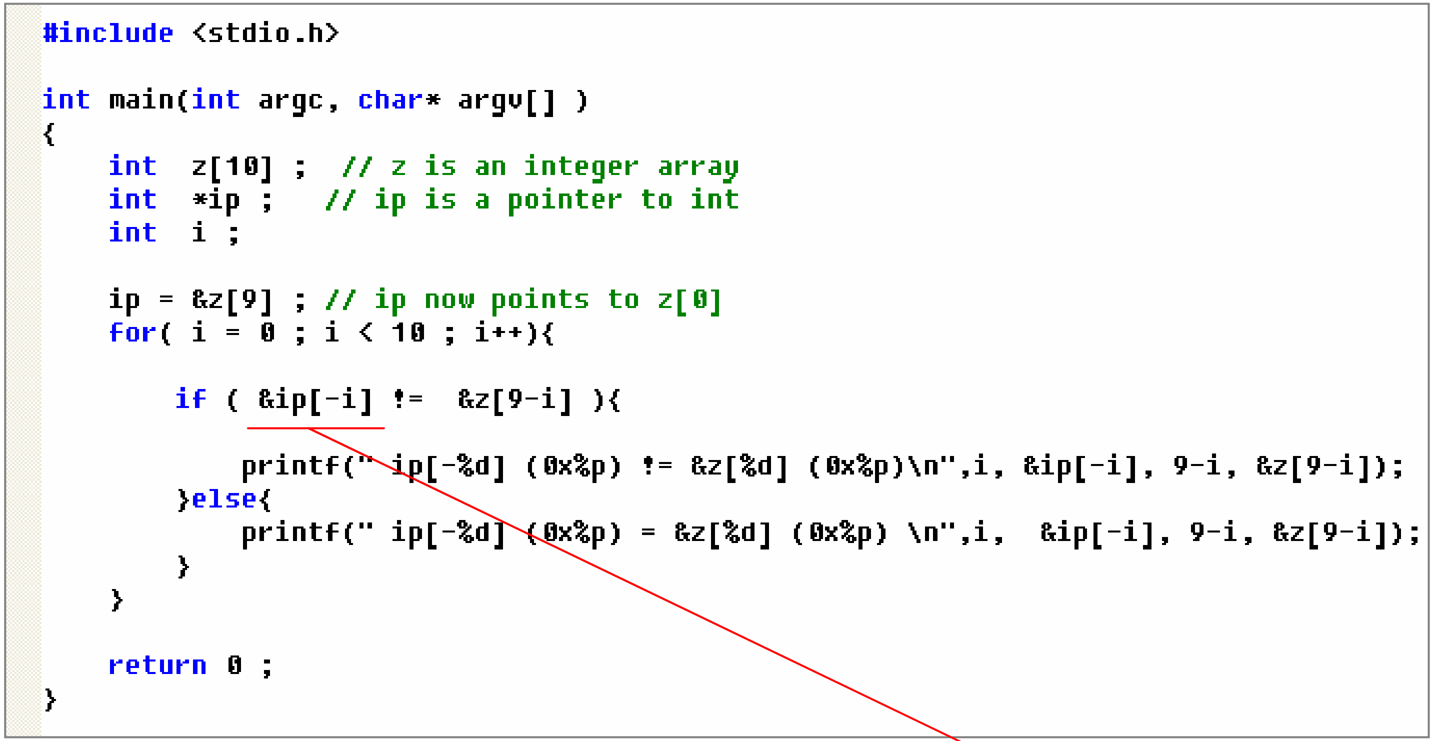
Casting to pointer pointing to char



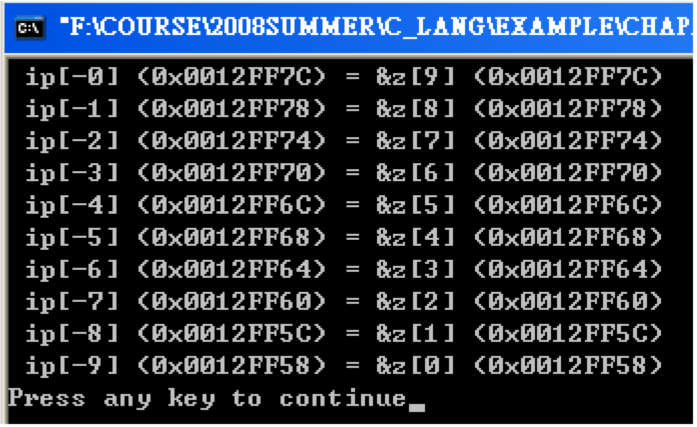
```
ip(0x0012FF80) - z(0x0012FF58) = 10
ip(0x0012FF80) - z(0x0012FF58) = 40
Press any key to continue_
```

Why not 0x28=40(decimal)?

Pointer and array   [4]



```c
#include <stdio.h>

int main(int argc, char* argv[] )
{
    int  z[10] ;  // z is an integer array
    int  *ip ;   // ip is a pointer to int
    int  i ;

    ip = &z[9] ; // ip now points to z[0]
    for( i = 0 ; i < 10 ; i++){

        if ( &ip[-i] !=  &z[9-i] ){

            printf(" ip[-%d] (0x%p) != &z[%d] (0x%p)\n",i, &ip[-i], 9-i, &z[9-i]);
        }else{
            printf(" ip[-%d] (0x%p) = &z[%d] (0x%p) \n",i,  &ip[-i], 9-i, &z[9-i]);
        }
    }

    return 0 ;
}
```



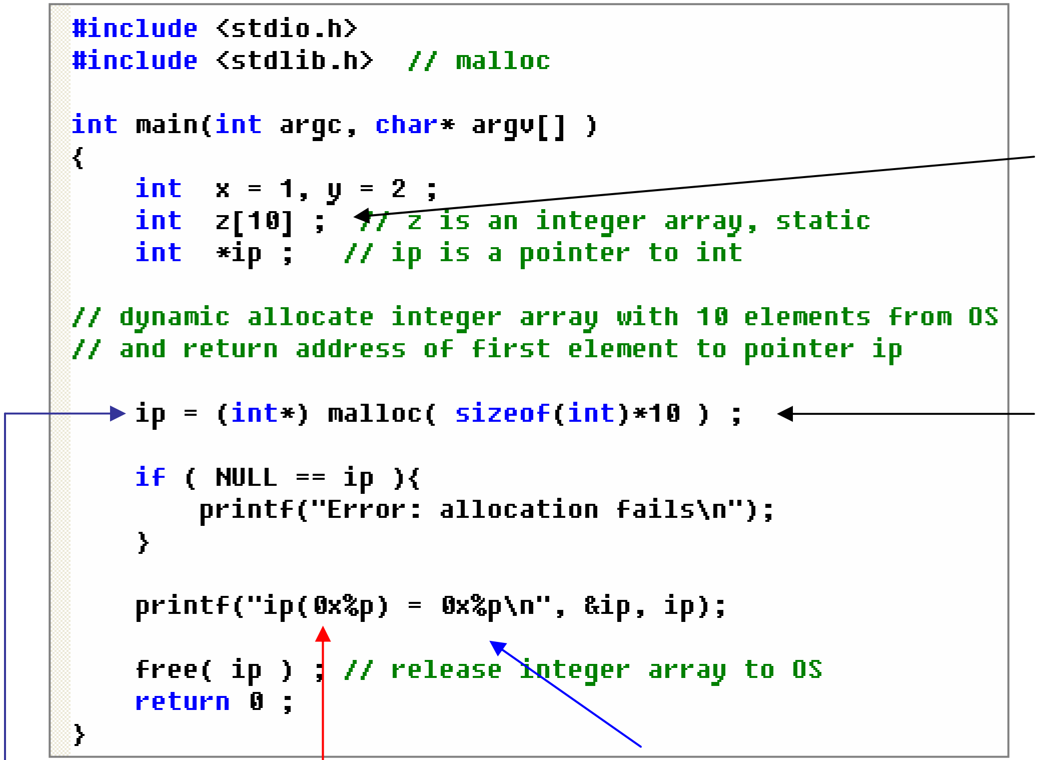
```
 ip[-0] (0x0012FF7C) = &z[9] (0x0012FF7C)
 ip[-1] (0x0012FF78) = &z[8] (0x0012FF78)
 ip[-2] (0x0012FF74) = &z[7] (0x0012FF74)
 ip[-3] (0x0012FF70) = &z[6] (0x0012FF70)
 ip[-4] (0x0012FF6C) = &z[5] (0x0012FF6C)
 ip[-5] (0x0012FF68) = &z[4] (0x0012FF68)
 ip[-6] (0x0012FF64) = &z[3] (0x0012FF64)
 ip[-7] (0x0012FF60) = &z[2] (0x0012FF60)
 ip[-8] (0x0012FF5C) = &z[1] (0x0012FF5C)
 ip[-9] (0x0012FF58) = &z[0] (0x0012FF58)
Press any key to continue_
```

&ip[-i] is equivalent to ip – i, C can accept array index smaller than zero

Address arithmetic

operation	Description
p++, p--	Increment (decrement) p to point to the next element, it is equivalent to p+=1 (p -=1)
p+i (p-i)	Point to i-th element beyond (in front of) p but value of p is fixed
p[i]	Equivalent to p + i
p + n (integer)	n must be an integer, its meaning is offset (偏移量)
p - q	Offset between pointer p and pointer q
p+q, p*q, p/q, p%q	invalid
Relational operator of two pointers p, q	valid, including p > q, p < q, p == q, p != q, p >= q, p <= q
malloc	Dynamic memory allocation
free	Release memory block which is allocated by *malloc*

Static and dynamic allocation



```c
#include <stdio.h>
#include <stdlib.h>  // malloc

int main(int argc, char* argv[] )
{
    int  x = 1, y = 2 ;
    int  z[10] ;  // z is an integer array, static
    int  *ip ;   // ip is a pointer to int

// dynamic allocate integer array with 10 elements from OS
// and return address of first element to pointer ip

    ip = (int*) malloc( sizeof(int)*10 ) ;

    if ( NULL == ip ){
        printf("Error: allocation fails\n");
    }

    printf("ip(0x%p) = 0x%p\n", &ip, ip);

    free( ip ) ; // release integer array to OS
    return 0 ;
}
```

z[10]是靜態陣列, 放在堆疊 (stack) 上

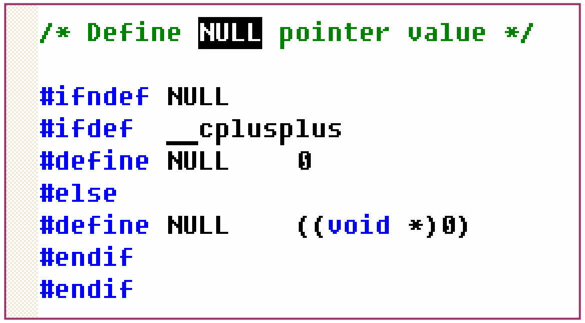
ip 指向一個動態陣列, 由作業系統從 heap 中截取 40 bytes

ip 所指向的動態陣列位址

ip本身的記憶體位址

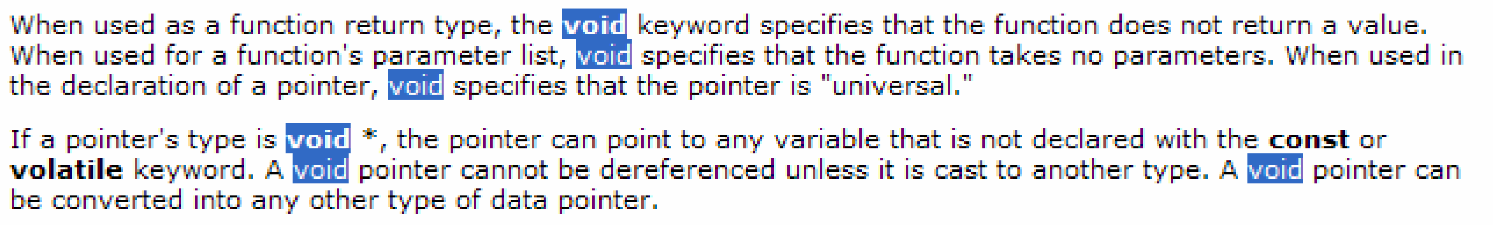
C guarantees that zero is never a valid address for data, so return value of zero can be used to signal an abnormal value

stdio.h



```c
/* Define NULL pointer value */

#ifndef NULL
#ifdef  __cplusplus
#define NULL    0
#else
#define NULL    ((void *)0)
#endif
#endif
```

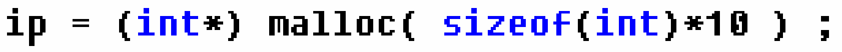
dynamic allocation, stdlib.h [1]



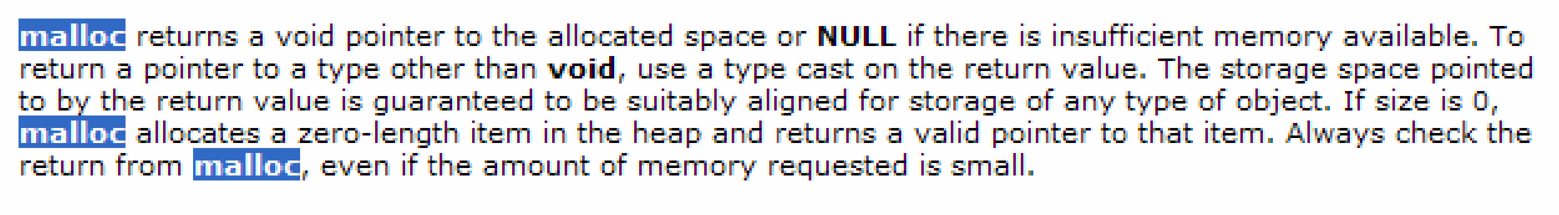
When used as a function return type, the void keyword specifies that the function does not return a value. When used for a function's parameter list, void specifies that the function takes no parameters. When used in the declaration of a pointer, void specifies that the pointer is "universal."

If a pointer's type is **void** *, the pointer can point to any variable that is not declared with the **const** or **volatile** keyword. A void pointer cannot be dereferenced unless it is cast to another type. A void pointer can be converted into any other type of data pointer.

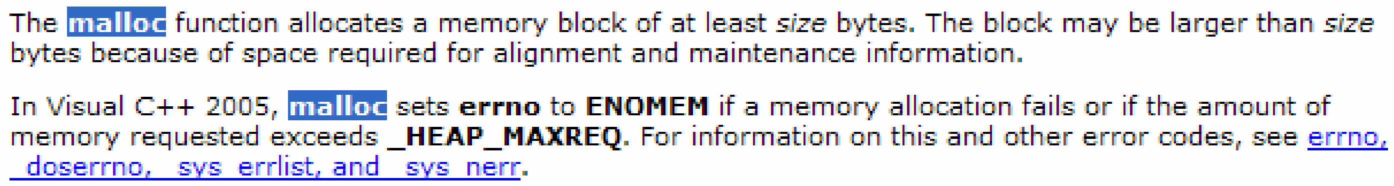
void * malloc(size_t size)



```
ip = (int*) malloc( sizeof(int)*10 ) ;
```



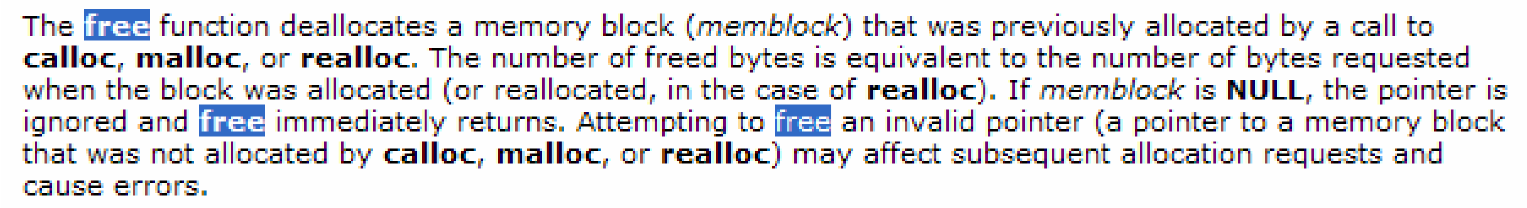
malloc returns a void pointer to the allocated space or **NULL** if there is insufficient memory available. To return a pointer to a type other than **void**, use a type cast on the return value. The storage space pointed to by the return value is guaranteed to be suitably aligned for storage of any type of object. If size is 0, malloc allocates a zero-length item in the heap and returns a valid pointer to that item. Always check the return from malloc, even if the amount of memory requested is small.



The malloc function allocates a memory block of at least *size* bytes. The block may be larger than *size* bytes because of space required for alignment and maintenance information.

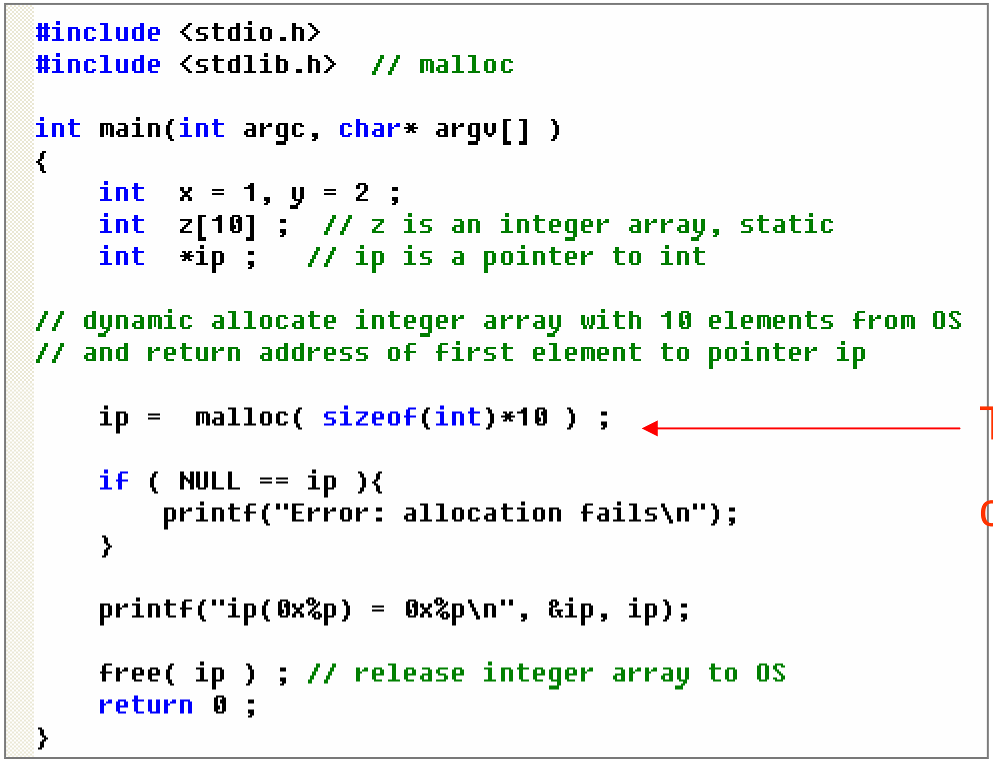
In Visual C++ 2005, malloc sets **errno** to **ENOMEM** if a memory allocation fails or if the amount of memory requested exceeds **_HEAP_MAXREQ**. For information on this and other error codes, see errno, _doserrno, _sys_errlist, and _sys_nerr.

void free(void *p)



The free function deallocates a memory block (*memblock*) that was previously allocated by a call to **calloc**, **malloc**, or **realloc**. The number of freed bytes is equivalent to the number of bytes requested when the block was allocated (or reallocated, in the case of **realloc**). If *memblock* is **NULL**, the pointer is ignored and free immediately returns. Attempting to free an invalid pointer (a pointer to a memory block that was not allocated by **calloc**, **malloc**, or **realloc**) may affect subsequent allocation requests and cause errors.

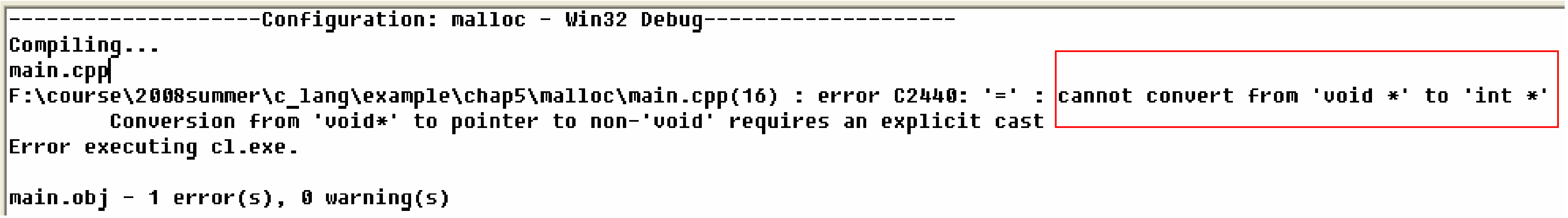
dynamic allocation, stdlib.h [2]



```
#include <stdio.h>
#include <stdlib.h>  // malloc

int main(int argc, char* argv[] )
{
    int  x = 1, y = 2 ;
    int  z[10] ;  // z is an integer array, static
    int  *ip ;   // ip is a pointer to int

// dynamic allocate integer array with 10 elements from OS
// and return address of first element to pointer ip

    ip =  malloc( sizeof(int)*10 ) ;

    if ( NULL == ip ){
        printf("Error: allocation fails\n");
    }

    printf("ip(0x%p) = 0x%p\n", &ip, ip);

    free( ip ) ; // release integer array to OS
    return 0 ;
}
```

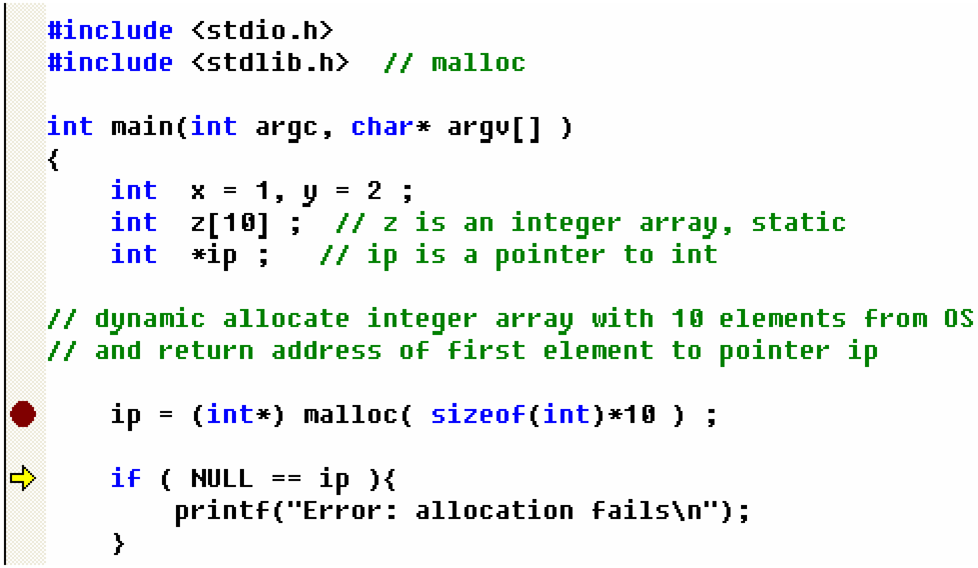
To allocate memory without casting

causes error of type checking

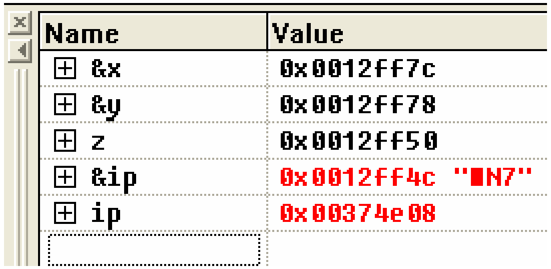


```
--------------------Configuration: malloc - Win32 Debug--------------------
Compiling...
main.cpp
F:\course\2008summer\c_lang\example\chap5\malloc\main.cpp(16) : error C2440: '=' : cannot convert from 'void *' to 'int *'
        Conversion from 'void*' to pointer to non-'void' requires an explicit cast
Error executing cl.exe.

main.obj - 1 error(s), 0 warning(s)
```

dynamic allocation [3]

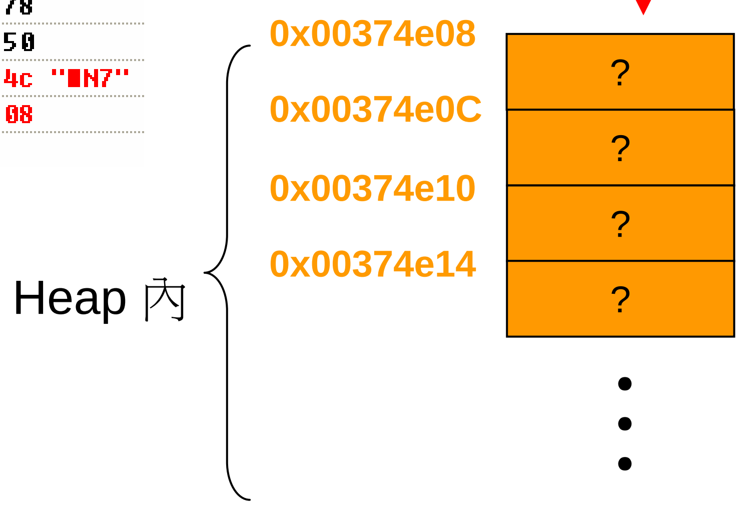


```
#include <stdio.h>
#include <stdlib.h>  // malloc

int main(int argc, char* argv[] )
{
    int  x = 1, y = 2 ;
    int  z[10] ;  // z is an integer array, static
    int  *ip ;   // ip is a pointer to int

// dynamic allocate integer array with 10 elements from OS
// and return address of first element to pointer ip

    ip = (int*) malloc( sizeof(int)*10 ) ;

    if ( NULL == ip ){
        printf("Error: allocation fails\n");
    }
```



Name	Value
⊞ &x	0x0012ff7c
⊞ &y	0x0012ff78
⊞ z	0x0012ff50
⊞ &ip	0x0012ff4c "■N7"
⊞ ip	0x00374e08

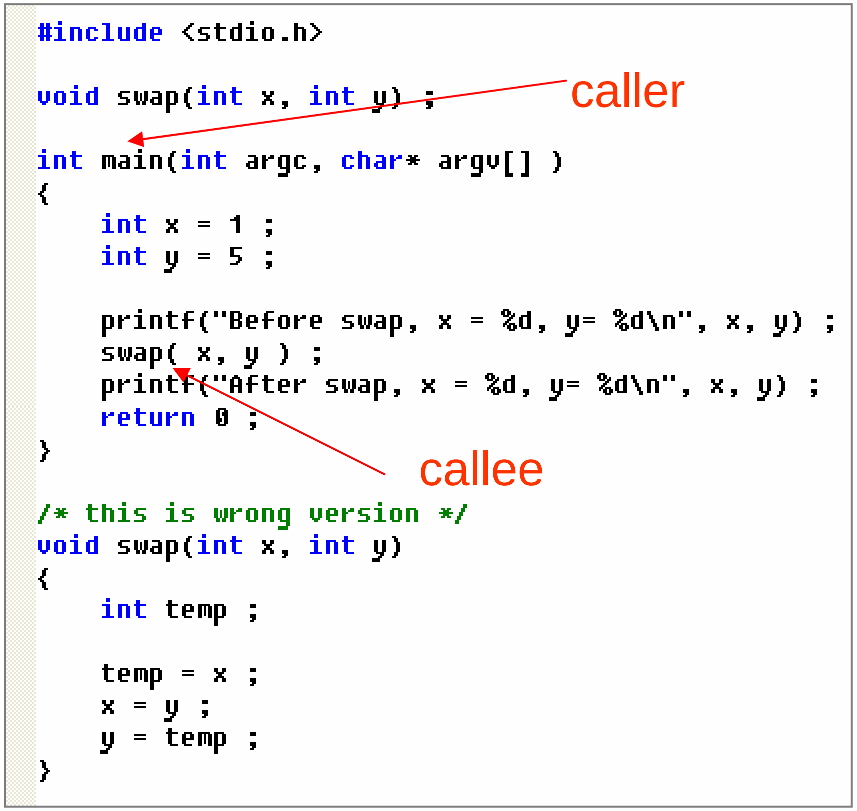
address	content	variable
0x0012FF4C	0x00374e08	ip
0x0012FF50	?	z[0]
0x0012FF54	?	z[1]
0x0012FF58	?	z[2]
0x0012FF5C	?	z[3]
0x0012FF60	?	z[4]
0x0012FF64	?	z[5]
0x0012FF68	?	z[6]
0x0012FF6C	?	z[7]
0x0012FF70	?	z[8]
0x0012FF74	?	z[9]
0x0012FF78	2	y
0x0012FF7C	1	x



OutLine

- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

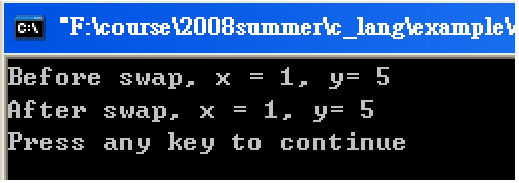
Call by value



```
#include <stdio.h>

void swap(int x, int y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
}
```

caller

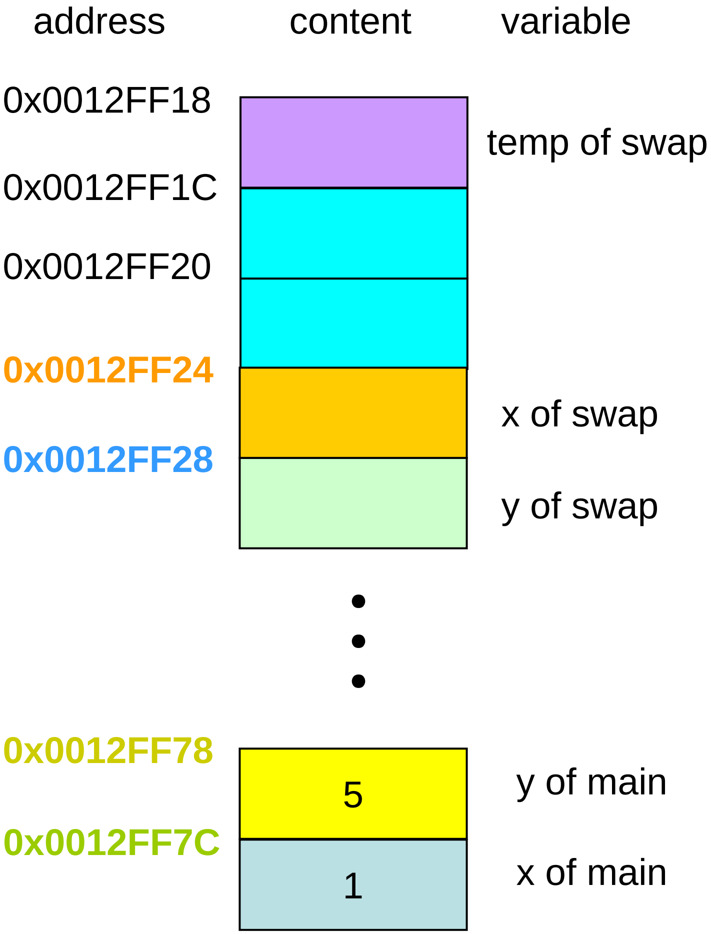
callee



"F:\course\2008summer\c_lang\example\

```
Before swap, x = 1, y= 5
After swap, x = 1, y= 5
Press any key to continue
```

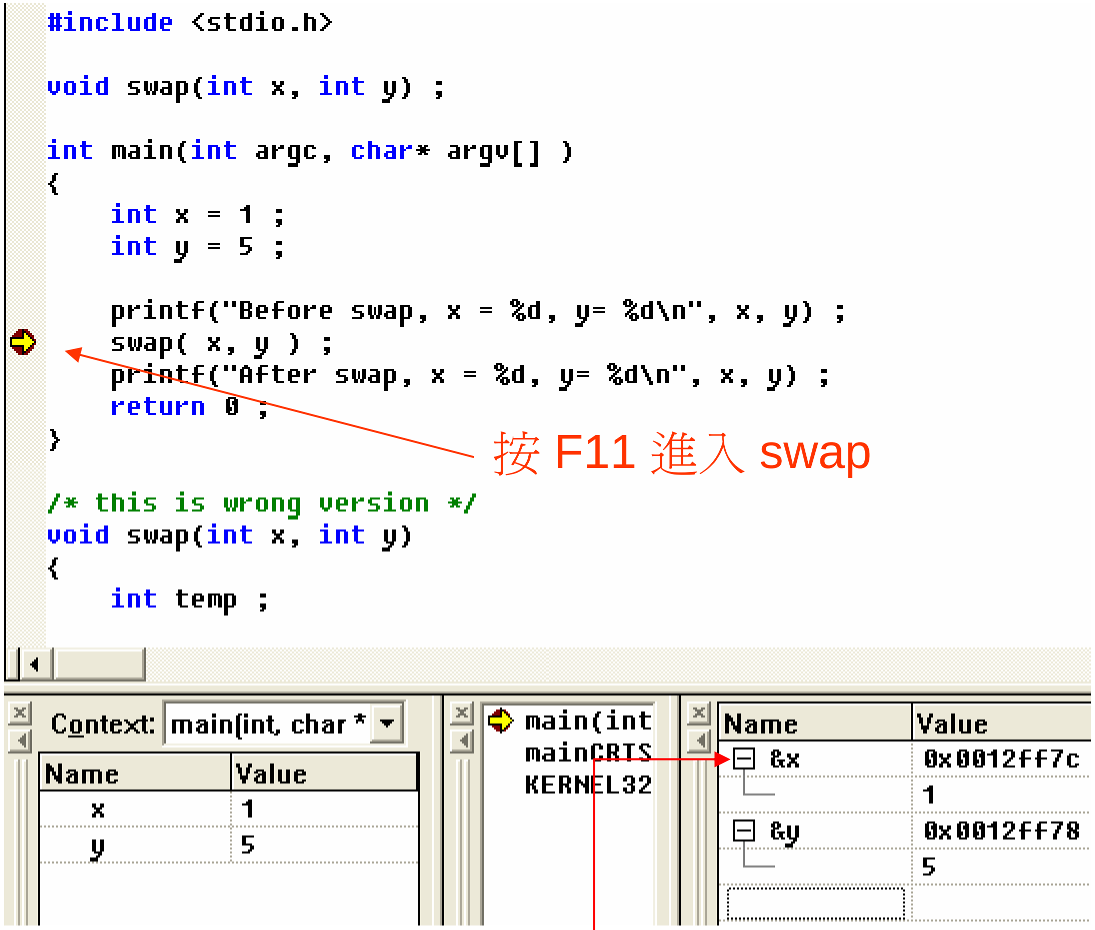
Question: why don’t x and y swap?



address	content	variable
0x0012FF18		temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24		x of swap
0x0012FF28		y of swap
⋮	⋮	⋮
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [1]

Use debugger



```
#include <stdio.h>

void swap(int x, int y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;
```

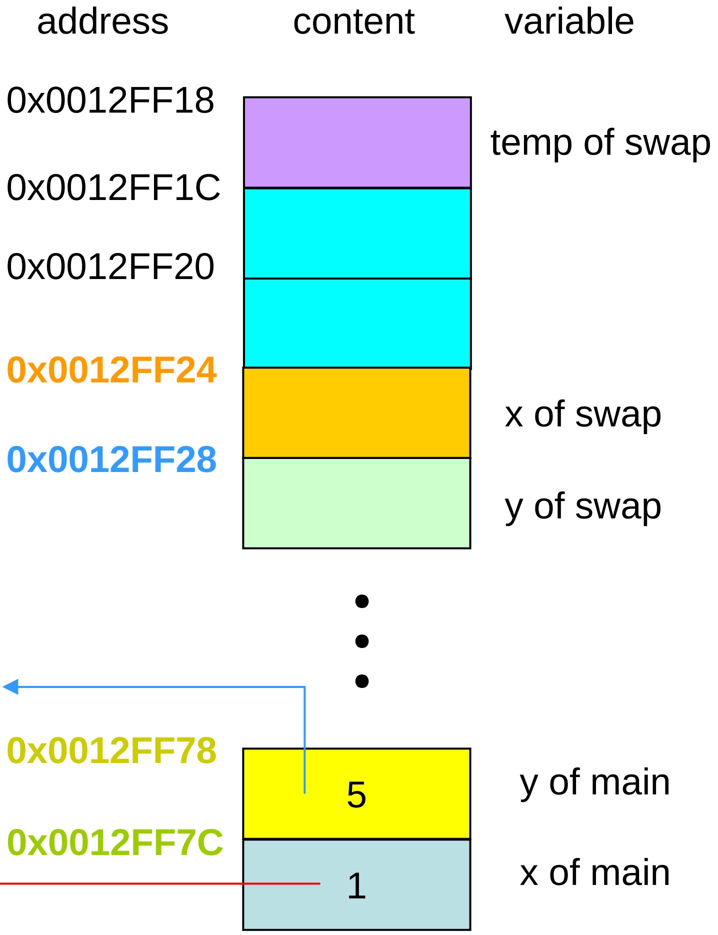
按 F11 進入 swap

Context: main(int, char *

Name	Value
x	1
y	5

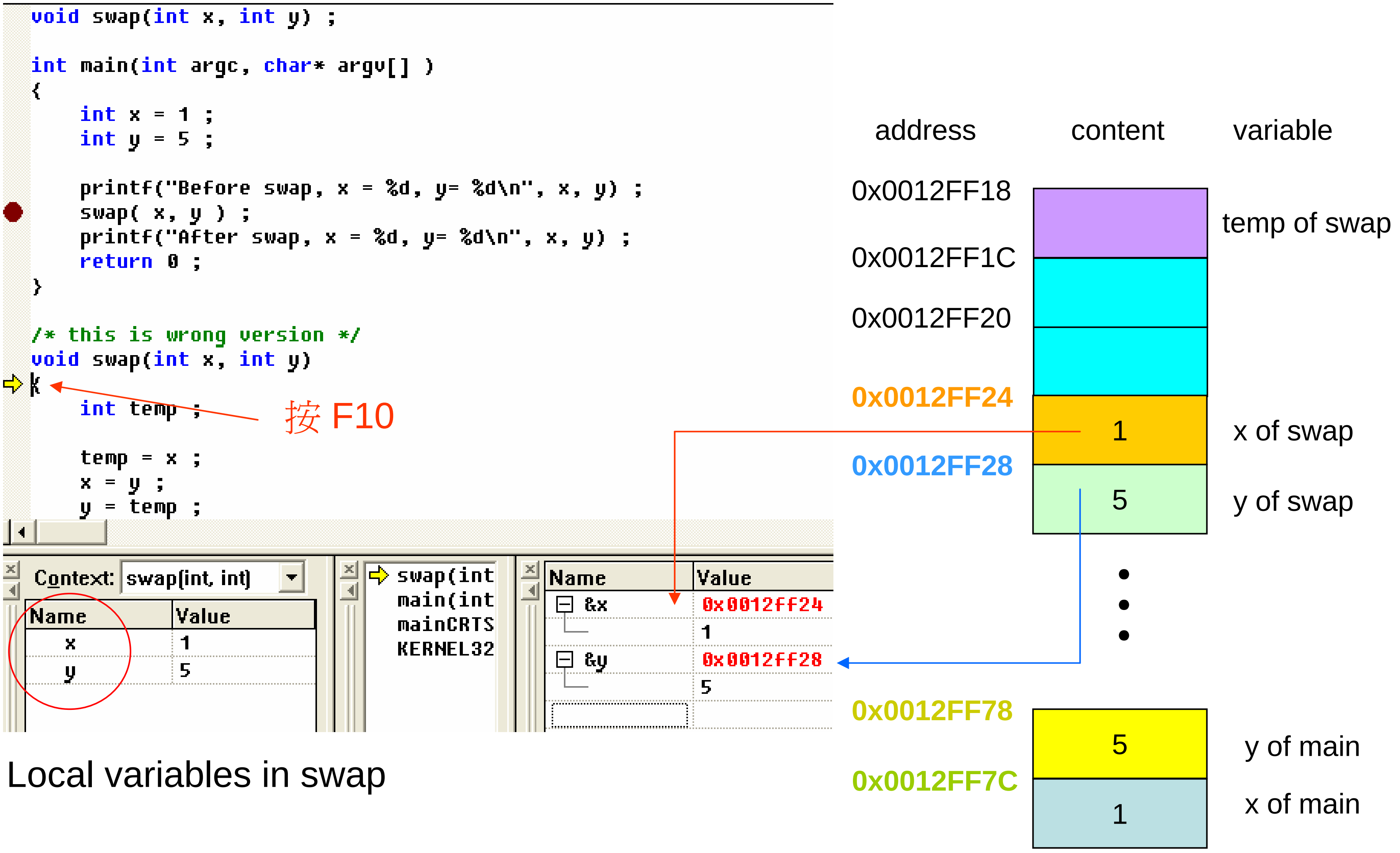
```
main(int
mainCRTS
KERNEL32
```

Name	Value
⊟ &x	0x0012ff7c
	1
⊟ &y	0x0012ff78
	5



address	content	variable
0x0012FF18		temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24		x of swap
0x0012FF28		y of swap
	• • •	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace: caller make a data copy to callee [2]



```
void swap(int x, int y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
```

按 F10

Context: swap(int, int)

Name	Value
x	1
y	5

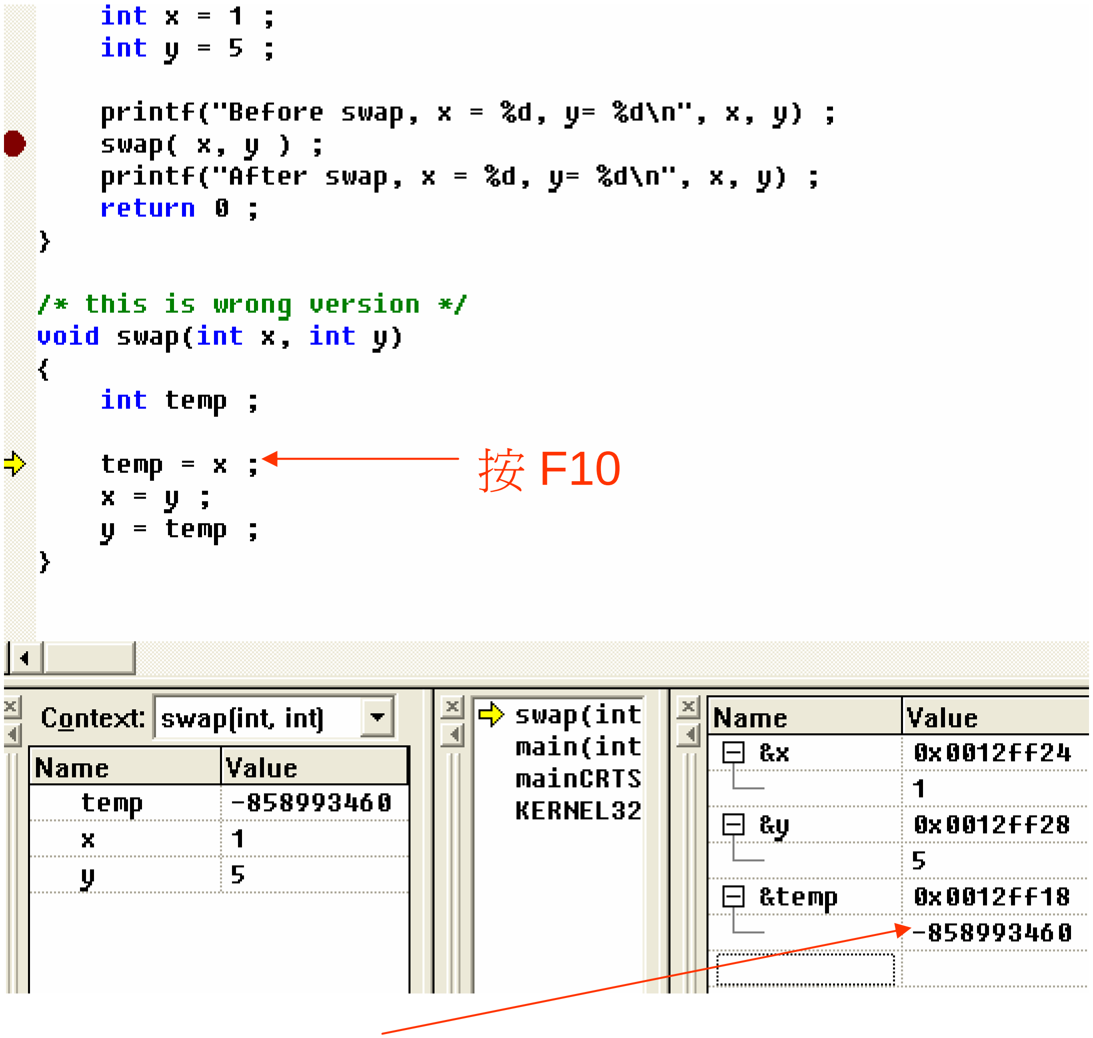
```
swap(int
main(int
mainCRTS
KERNEL32
```

Name	Value
⊟ &x	0x0012ff24
	1
⊟ &y	0x0012ff28
	5

Local variables in swap

address	content	variable
0x0012FF18		temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	1	x of swap
0x0012FF28	5	y of swap
• • •		
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [3]



```
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
}
```

按 F10

Context: swap(int, int)

Name	Value
temp	-858993460
x	1
y	5

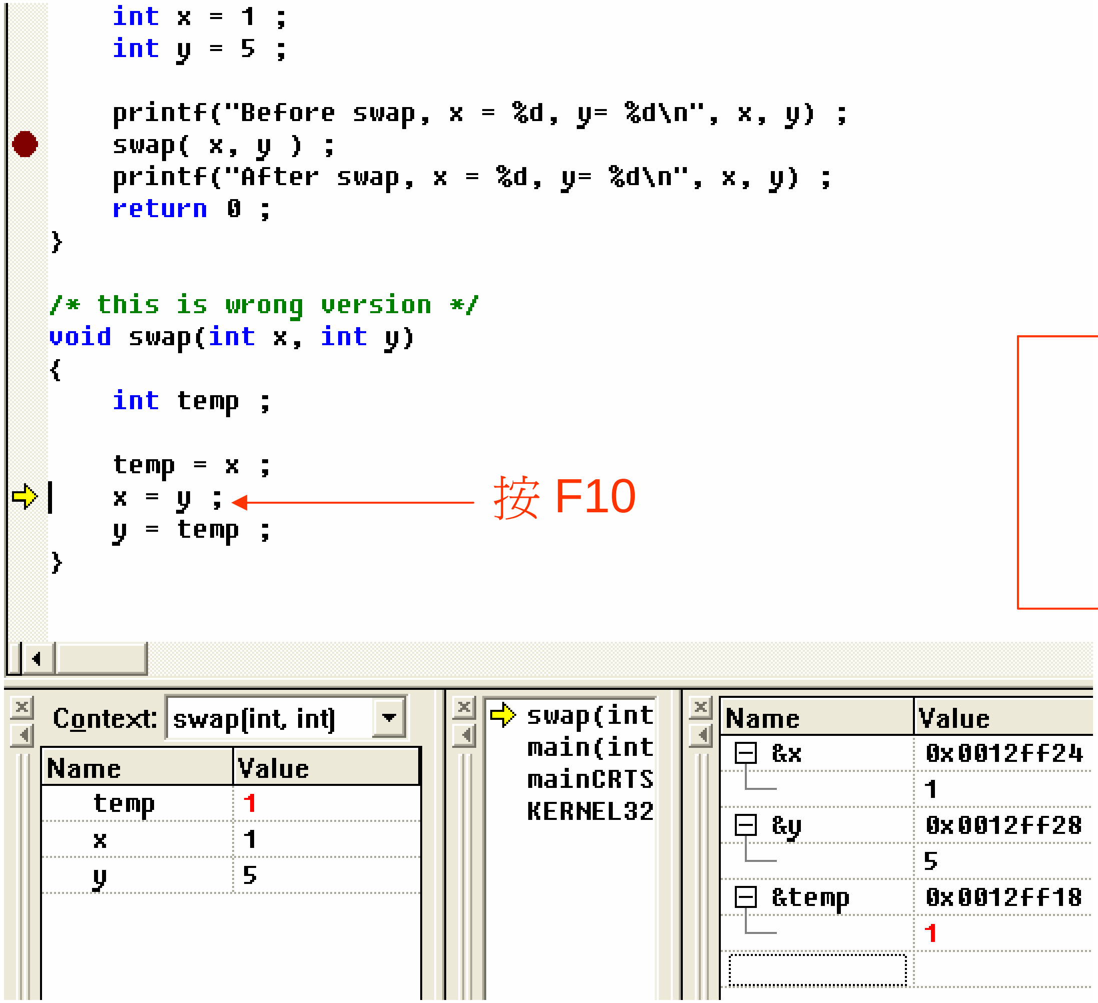
```
swap(int
main(int
mainCRTS
KERNEL32
```

Name	Value
⊟ &x	0x0012ff24
	1
⊟ &y	0x0012ff28
	5
⊟ &temp	0x0012ff18
	-858993460

temp has meaningless content

address	content	variable
0x0012FF18	-858993460	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	1	x of swap
0x0012FF28	5	y of swap
	• • •	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [4]



```
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
}
```

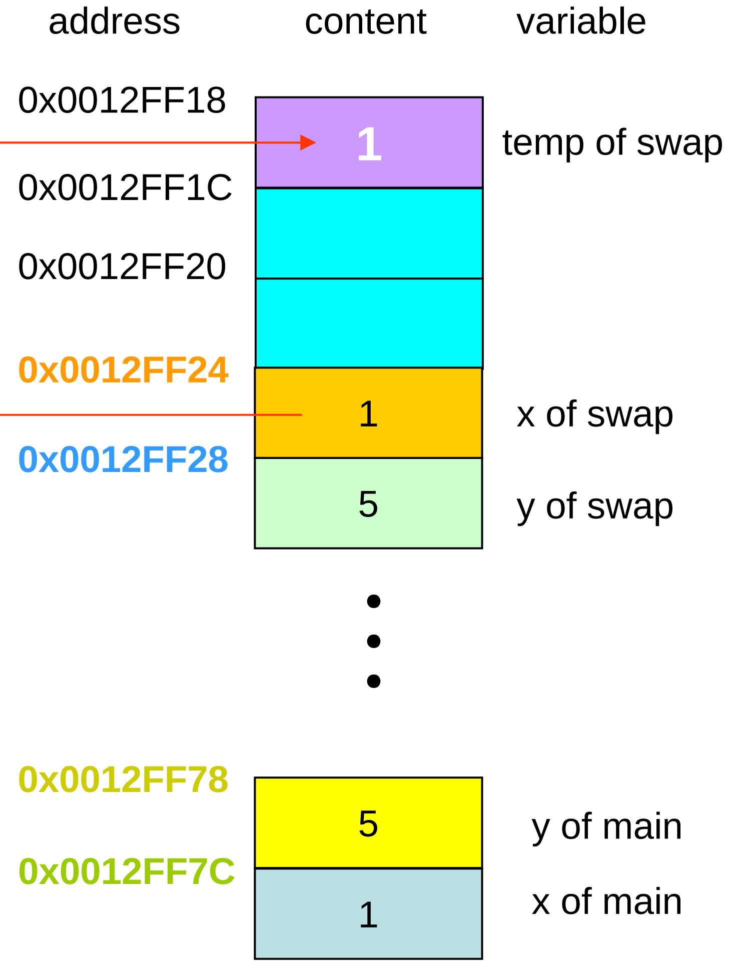
按 F10

Context: swap(int, int)

Name	Value
temp	1
x	1
y	5

```
swap(int
main(int
mainCRTS
KERNEL32
```

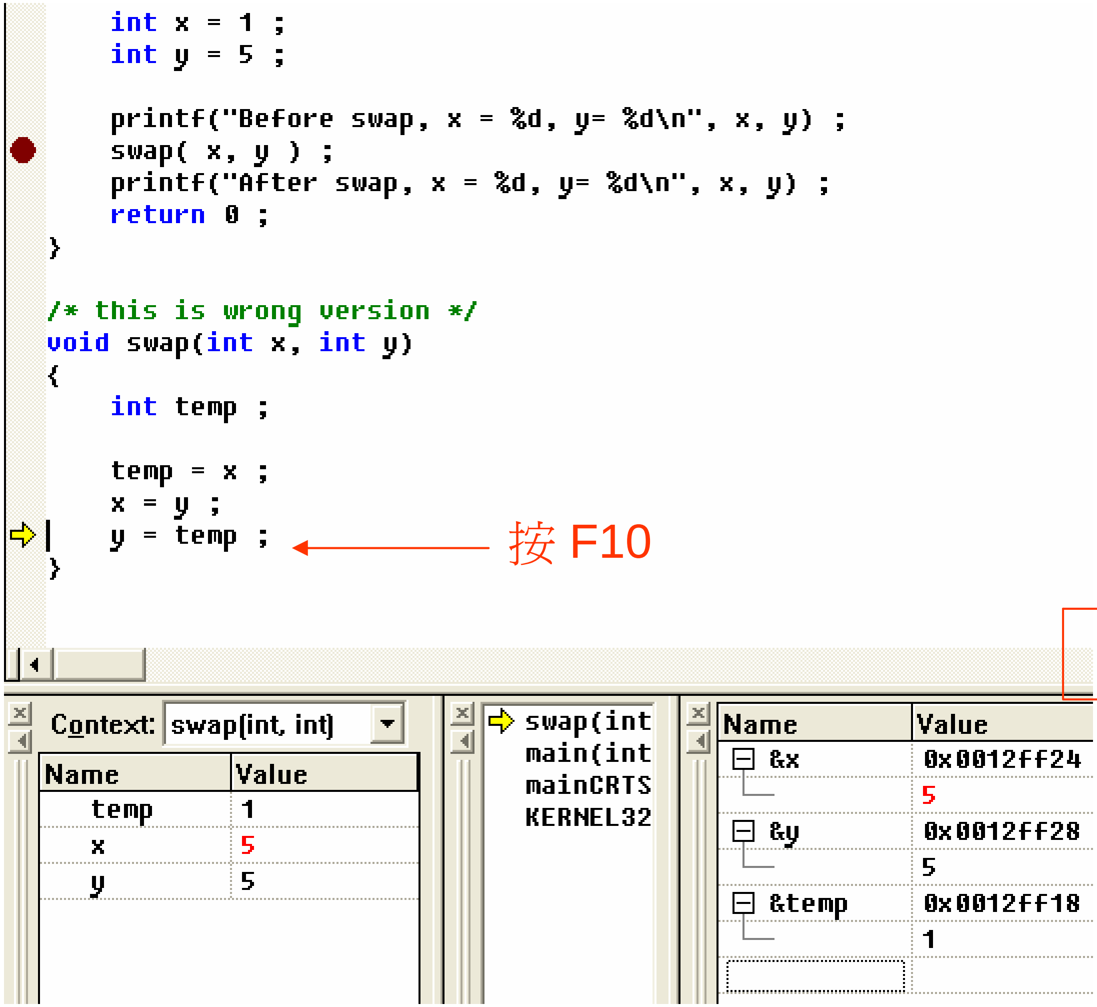
Name	Value
⊟ &x	0x0012ff24
	1
⊟ &y	0x0012ff28
	5
⊟ &temp	0x0012ff18
	1



address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	1	x of swap
0x0012FF28	5	y of swap
	• • •	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

move value of x of *swap* to temp of *swap*

Call graph trace [5]



```
    int x = 1 ;
    int y = 5 ;

    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
}
```

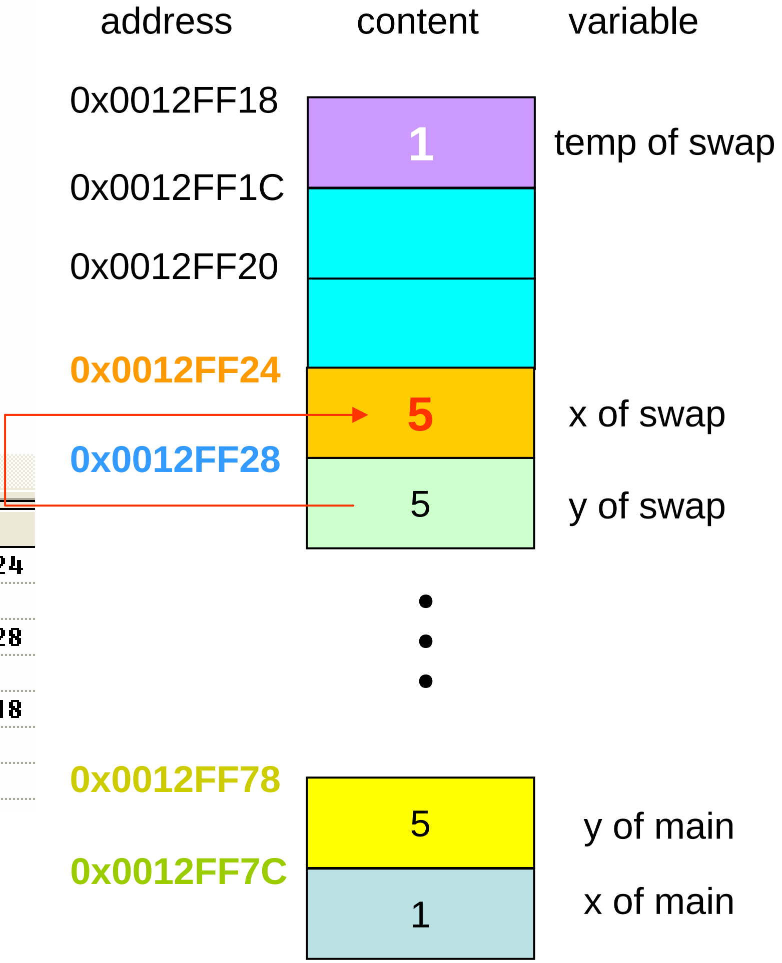
按 F10

Context: swap(int, int)

Name	Value
temp	1
x	5
y	5

```
swap(int
main(int
mainCRTS
KERNEL32
```

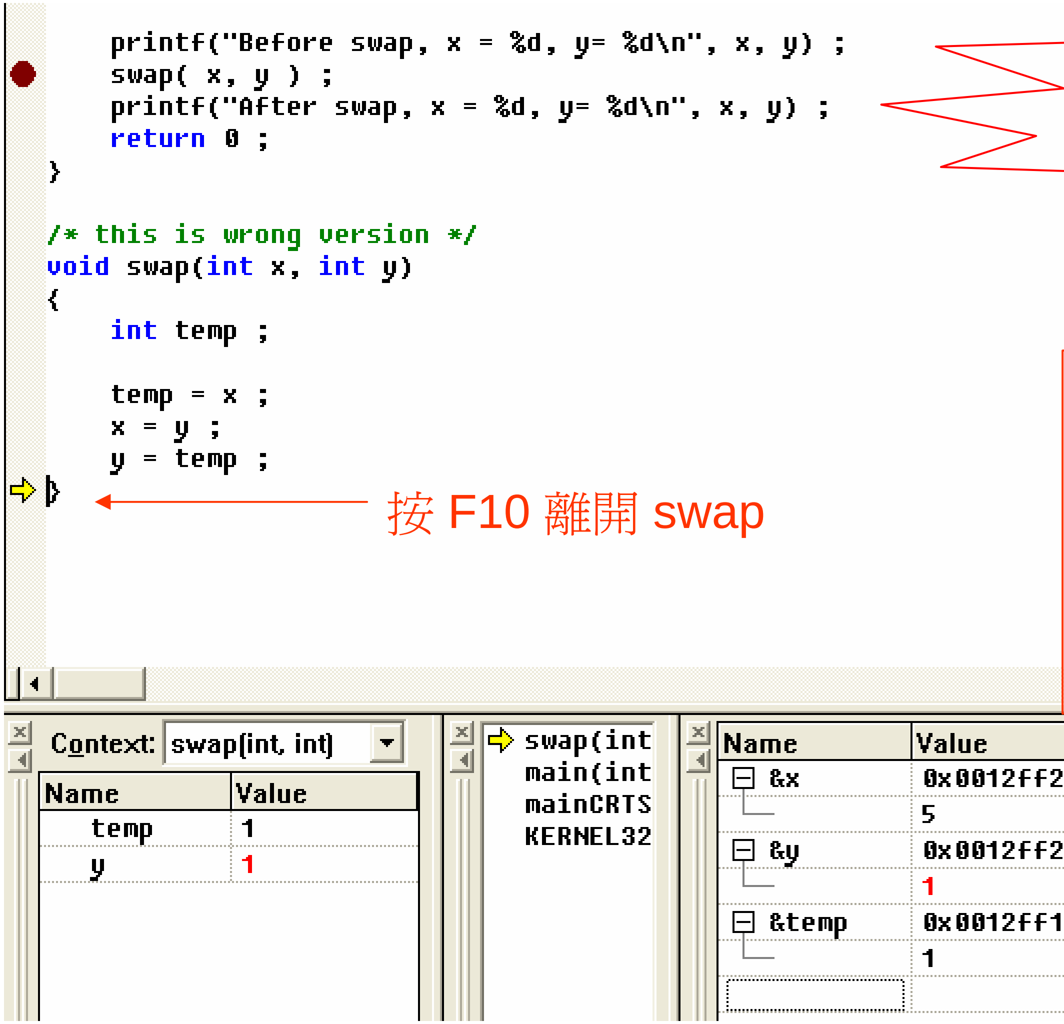
Name	Value
&x	0x0012ff24
	5
&y	0x0012ff28
	5
&temp	0x0012ff18
	1



address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	5	x of swap
0x0012FF28	5	y of swap
	⋮	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

move value of y of *swap* to x of *swap*

Call graph trace [6]



```
    printf("Before swap, x = %d, y= %d\n", x, y) ;
    swap( x, y ) ;
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

/* this is wrong version */
void swap(int x, int y)
{
    int temp ;

    temp = x ;
    x = y ;
    y = temp ;
}
```

按 F10 離開 swap

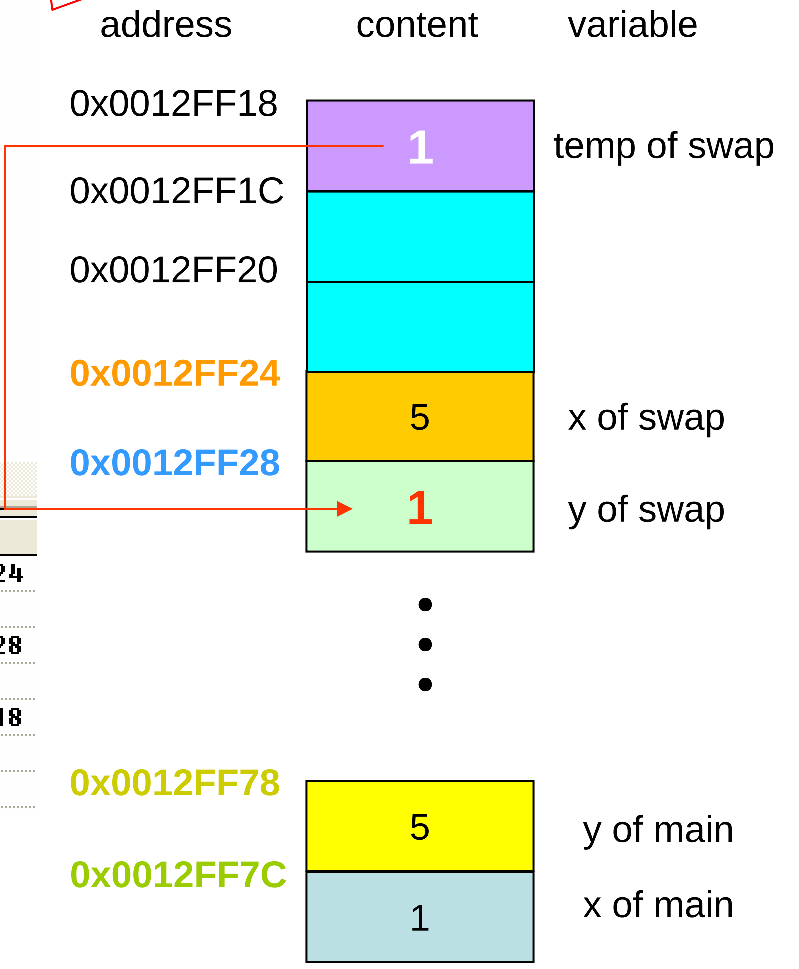
Context: swap(int, int)

Name	Value
temp	1
y	1

```
swap(int
main(int
mainCRTS
KERNEL32
```

Name	Value
⊟ &x	0x0012ff24
	5
⊟ &y	0x0012ff28
	1
⊟ &temp	0x0012ff18
	1

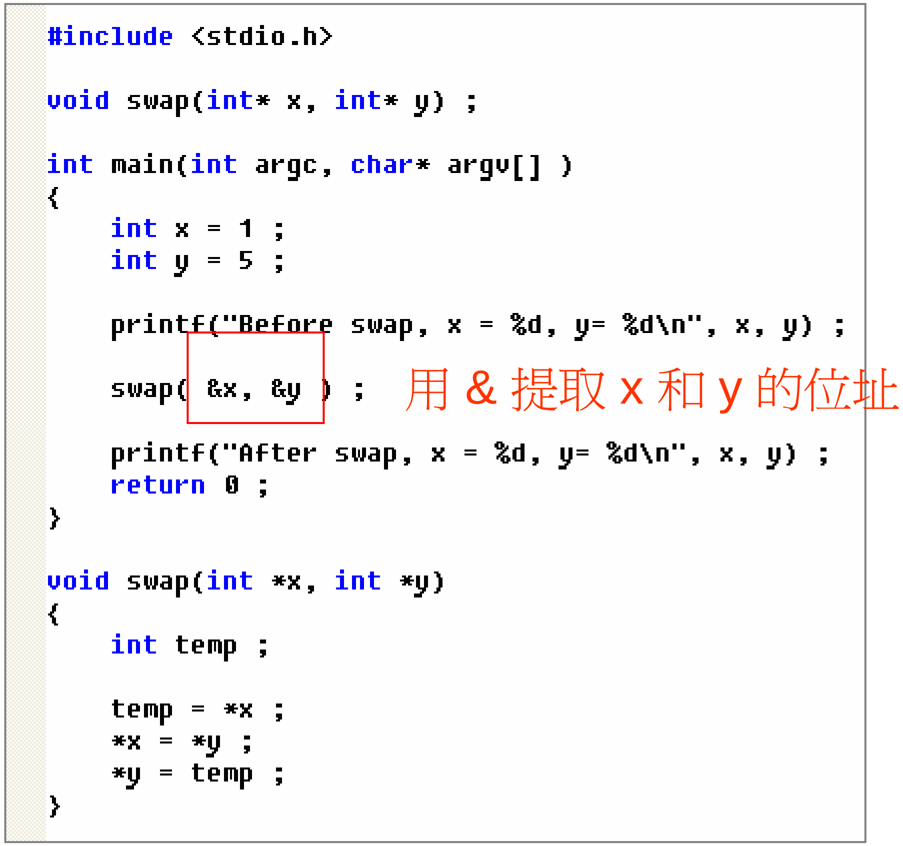
Only swap x, y in *swap*, not in *main*



address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	5	x of swap
0x0012FF28	1	y of swap
• • •		
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

move value of temp of *swap* to y of *swap*

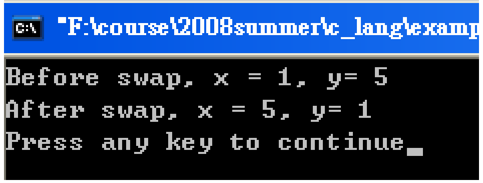
Call by value – pointer



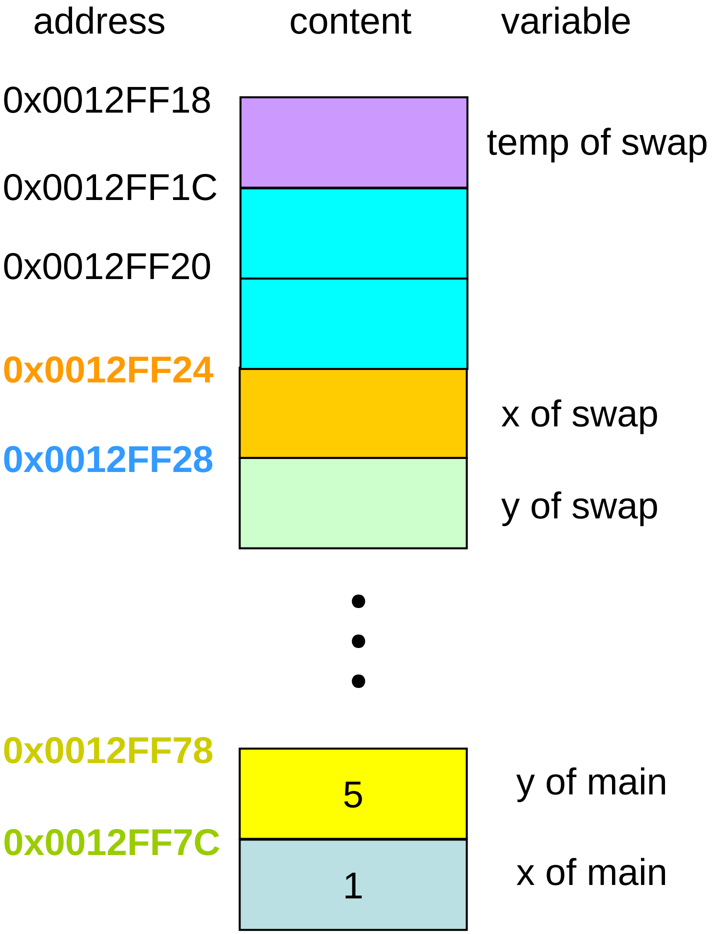
```
#include <stdio.h>

void swap(int* x, int* y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

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

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

用 & 提取 x 和 y 的位址

swap x and y indeed

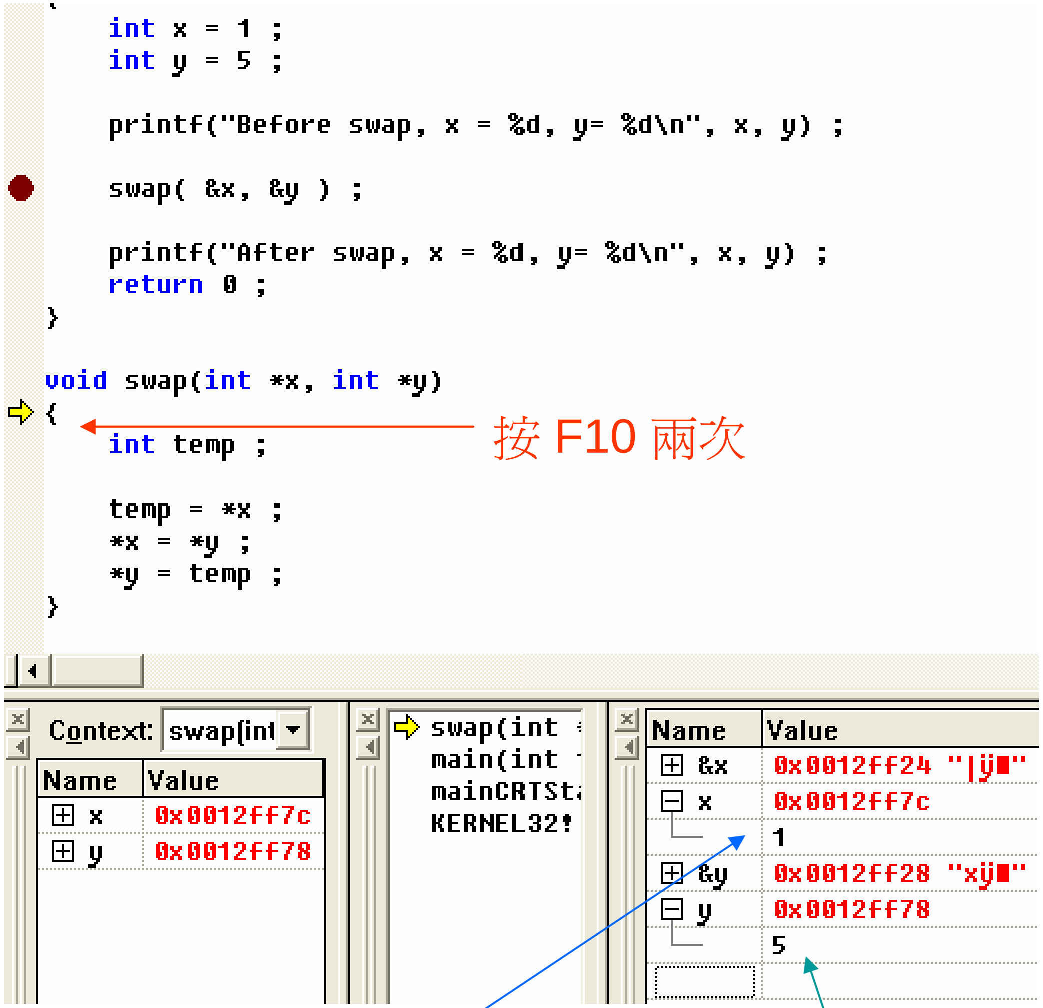


```
Before swap, x = 1, y= 5
After swap, x = 5, y= 1
Press any key to continue_
```



address	content	variable
0x0012FF18		temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24		x of swap
0x0012FF28		y of swap
	• • •	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [1]



```
    int x = 1 ;
    int y = 5 ;

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

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

按 F10 兩次

Context: swap(int

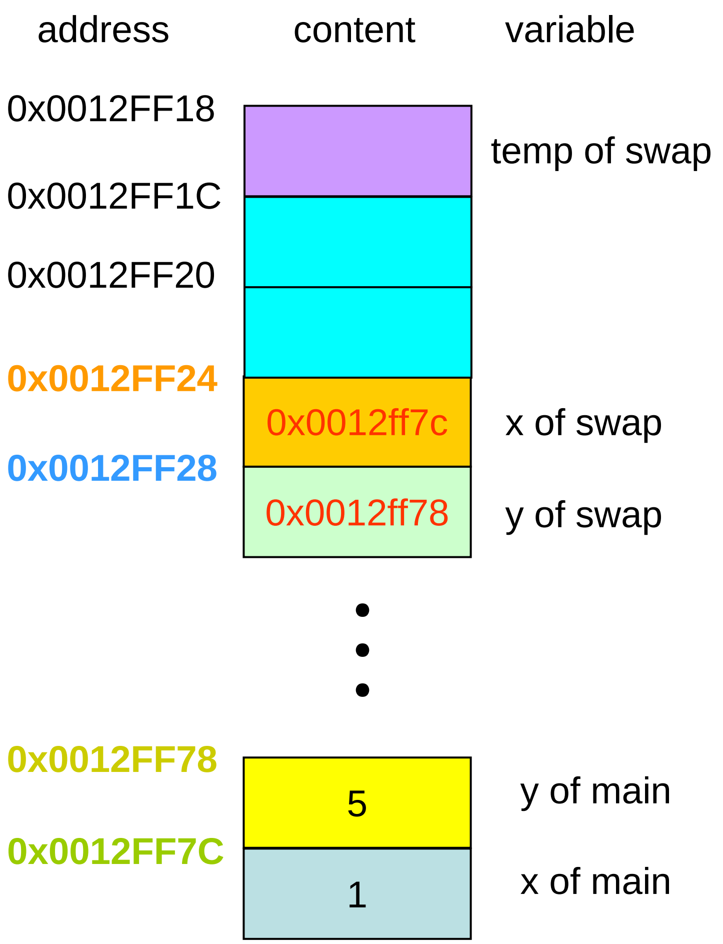
Name	Value
⊞ x	0x0012ff7c
⊞ y	0x0012ff78

```
swap(int
main(int
mainCRTSta
KERNEL32!
```

Name	Value
⊞ &x	0x0012ff24 "|ÿ■"
⊟ x	0x0012ff7c
	1
⊞ &y	0x0012ff28 "xÿ■"
⊟ y	0x0012ff78
	5

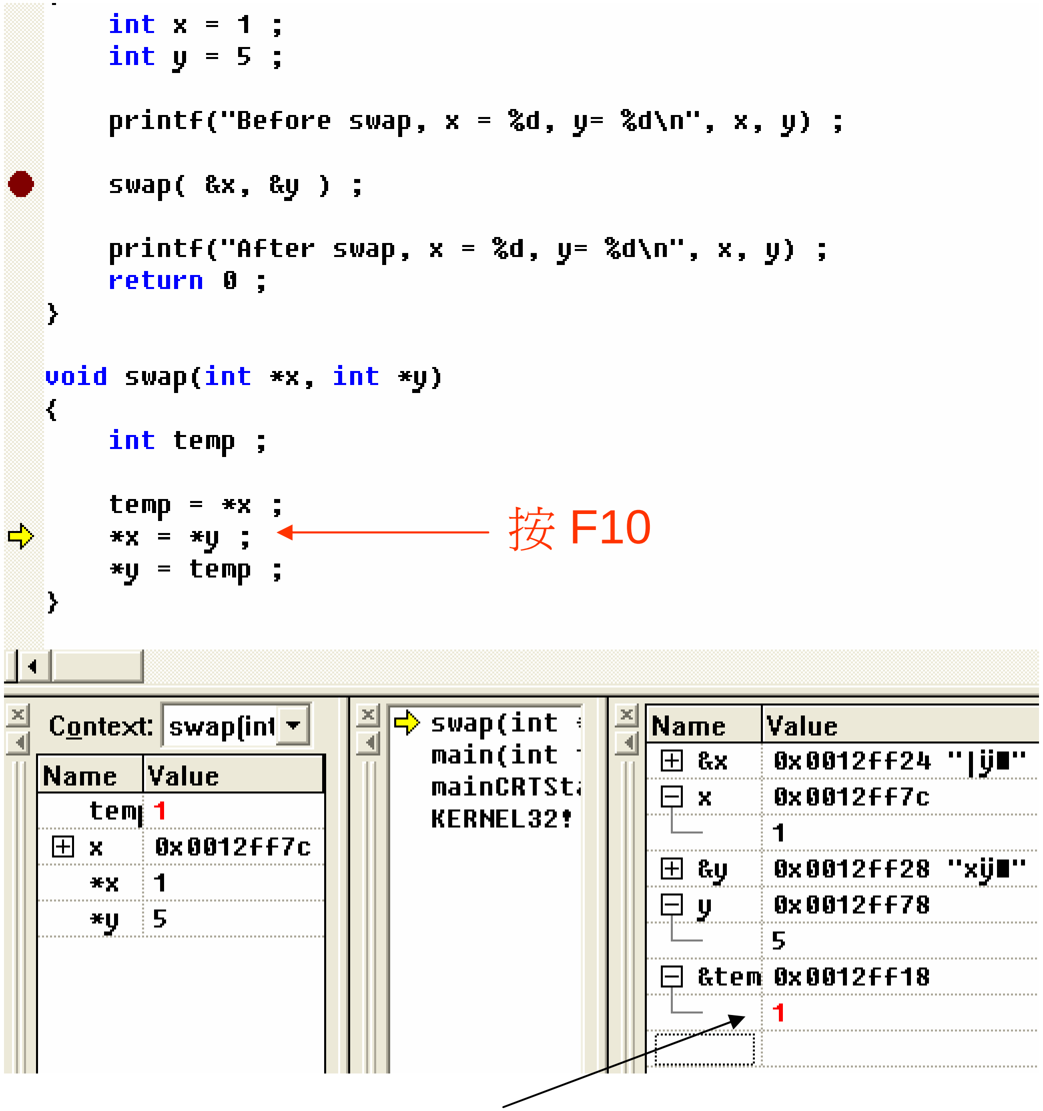
value of object which x points to

value of object which y points to



address	content	variable
0x0012FF18		temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	⋮	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [2]



```
    int x = 1 ;
    int y = 5 ;

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

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

按 F10

Context: swap(int

Name	Value
temp	1
⊞ x	0x0012ff7c
*x	1
*y	5

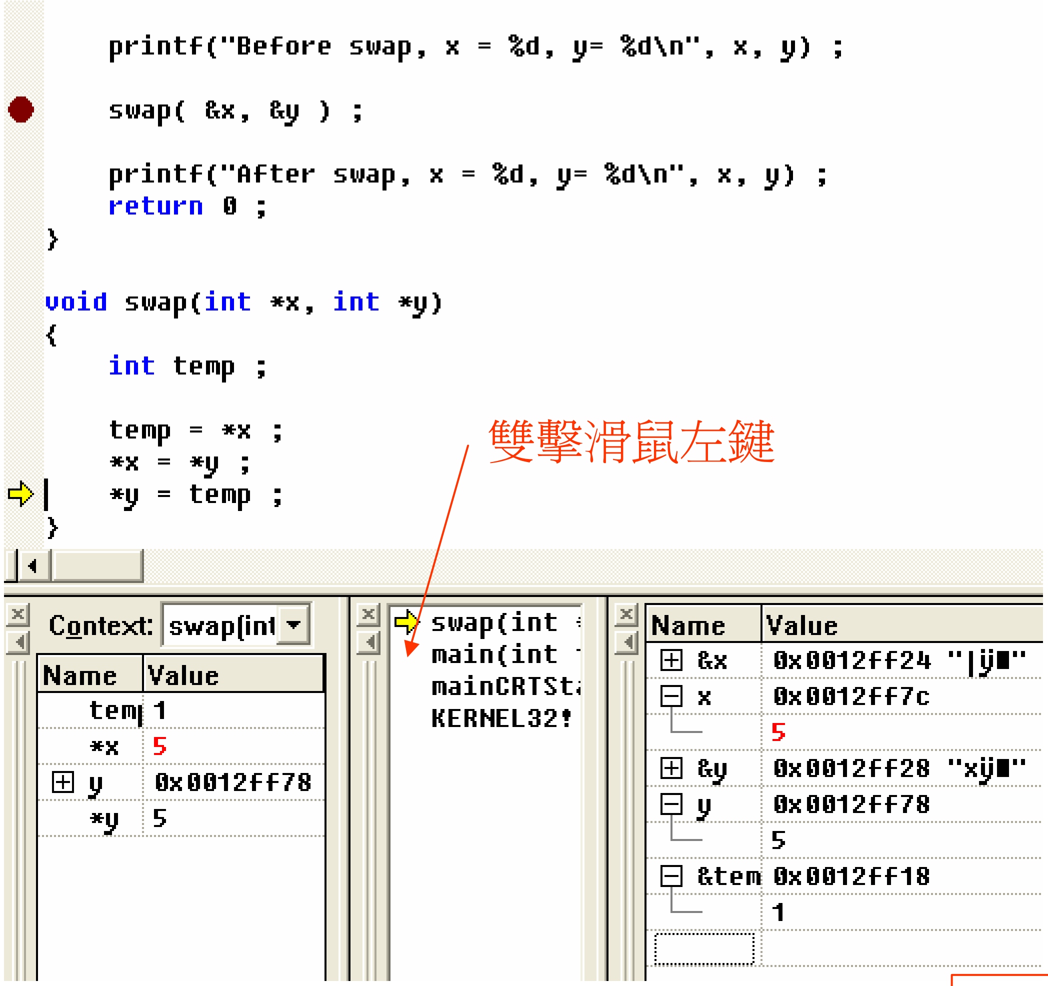
```
swap(int
main(int
mainCRTSt
KERNEL32!
```

Name	Value
⊞ &x	0x0012ff24 "|ÿ■"
⊟ x	0x0012ff7c
	1
⊞ &y	0x0012ff28 "xÿ■"
⊟ y	0x0012ff78
	5
⊟ &tem	0x0012ff18
	1

move value of x of main to temp of swap

address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	⋮	
0x0012FF78	5	y of main
0x0012FF7C	1	x of main

Call graph trace [3]



```
    printf("Before swap, x = %d, y= %d\n", x, y) ;

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

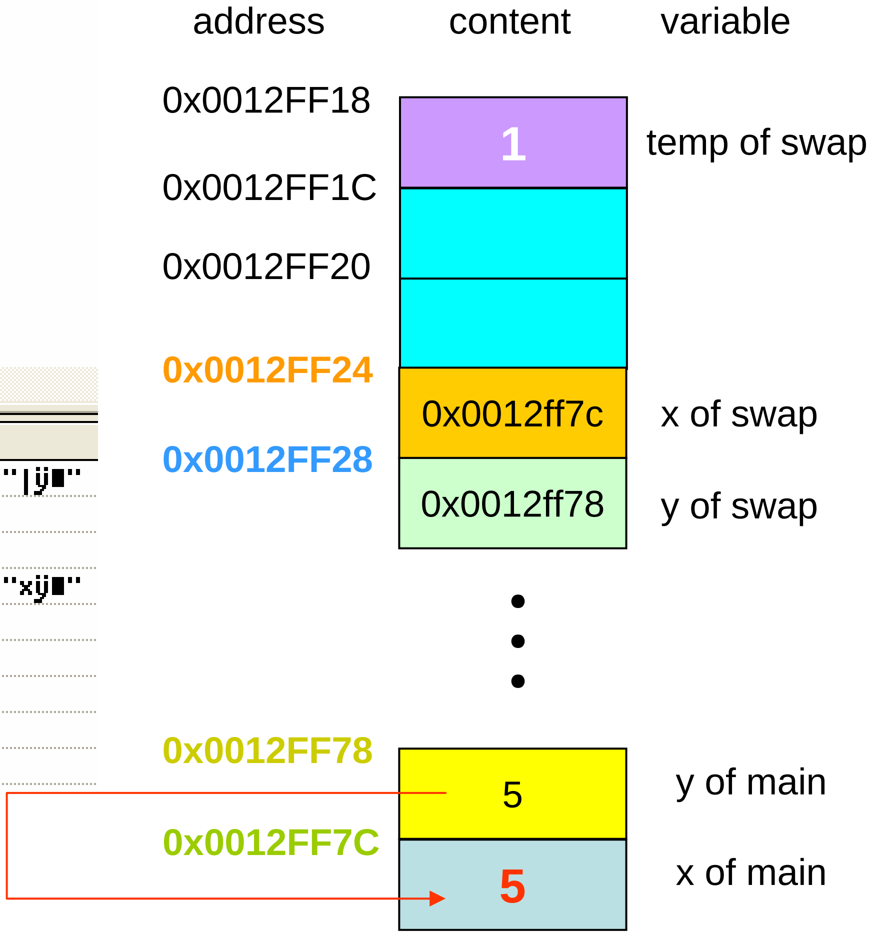
雙擊滑鼠左鍵

Context: swap(int

Name	Value
temp	1
*x	5
⊞ y	0x0012ff78
*y	5

```
swap(int
main(int
mainCRTSta
KERNEL32!
```

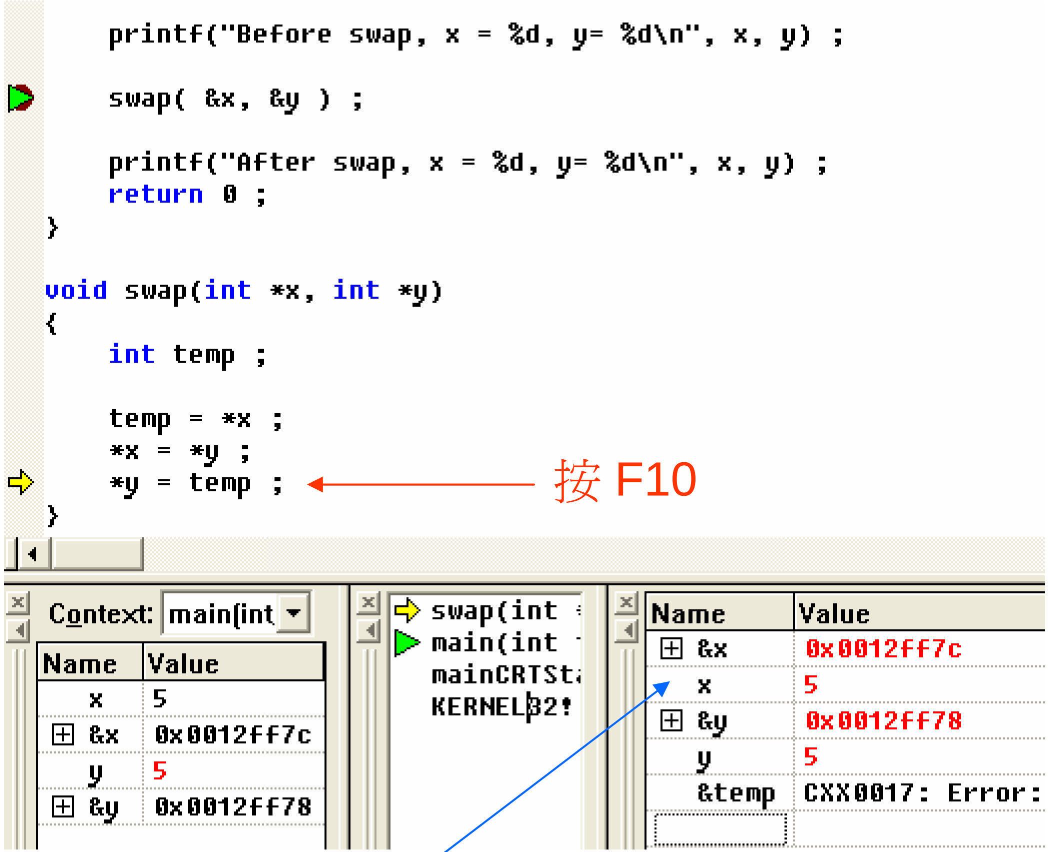
Name	Value
⊞ &x	0x0012ff24 "|ÿ■"
⊟ x	0x0012ff7c
	5
⊞ &y	0x0012ff28 "xÿ■"
⊟ y	0x0012ff78
	5
⊟ &tem	0x0012ff18
	1



address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	• • •	
0x0012FF78	5	y of main
0x0012FF7C	5	x of main

Question: how to check content of x in main ?

Call graph trace [4]



```
    printf("Before swap, x = %d, y= %d\n", x, y) ;

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

按 F10

Context: main(int

Name	Value
x	5
⊞ &x	0x0012ff7c
y	5
⊞ &y	0x0012ff78

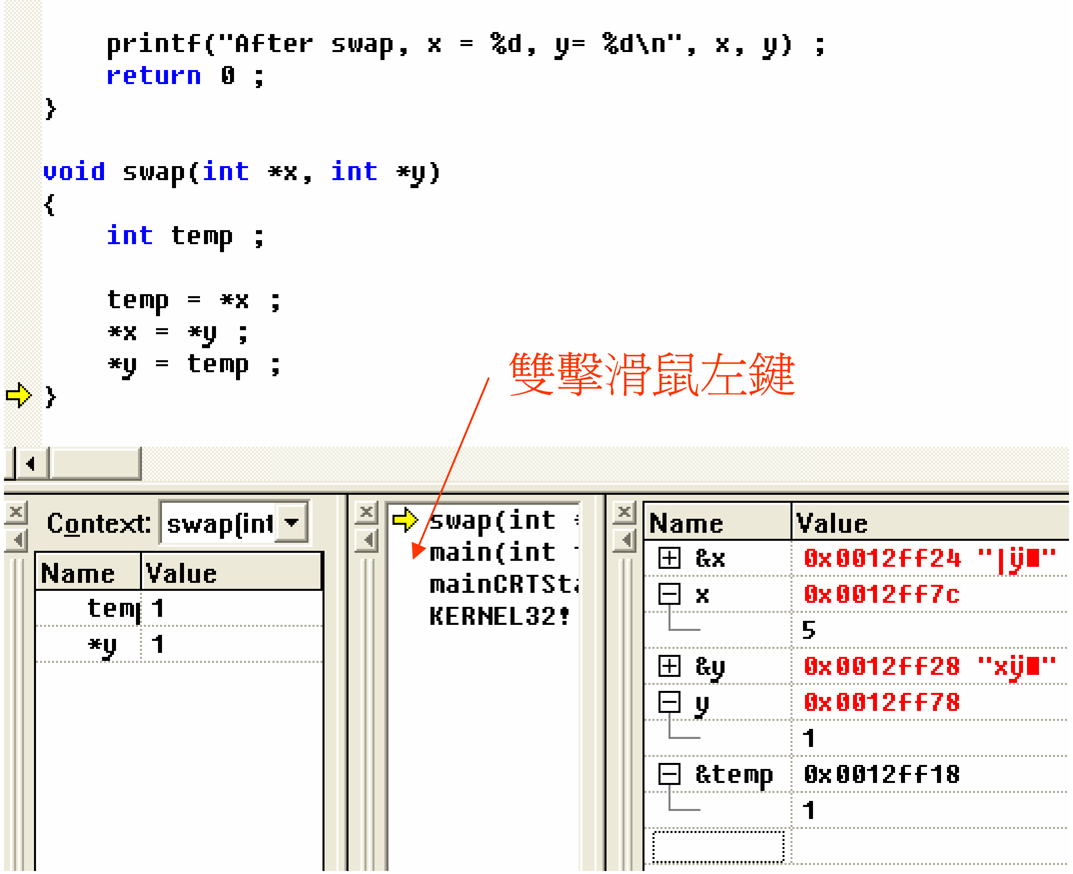
```
swap(int
main(int
mainCRTSt
KERNEL32!
```

Name	Value
⊞ &x	0x0012ff7c
x	5
⊞ &y	0x0012ff78
y	5
&temp	CXX0017: Error:

x of main, check its address

address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	⋮	
0x0012FF78	5	y of main
0x0012FF7C	5	x of main

Call graph trace [5]



```
    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

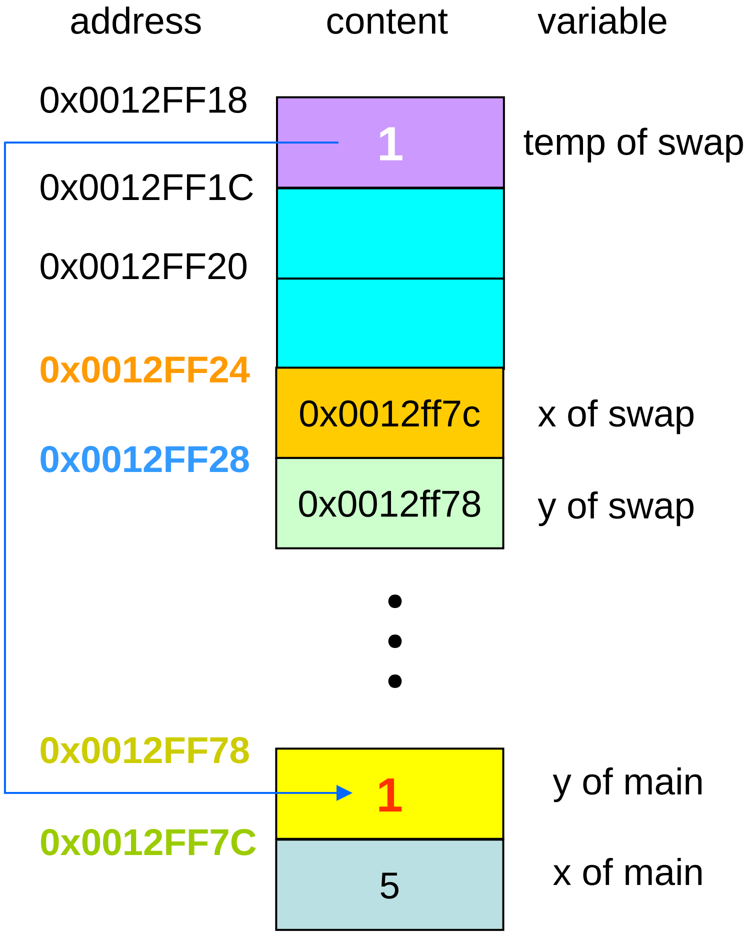
雙擊滑鼠左鍵

Context: swap(int

Name	Value
temp	1
*y	1

```
swap(int
main(int
mainCRTSt
KERNEL32!
```

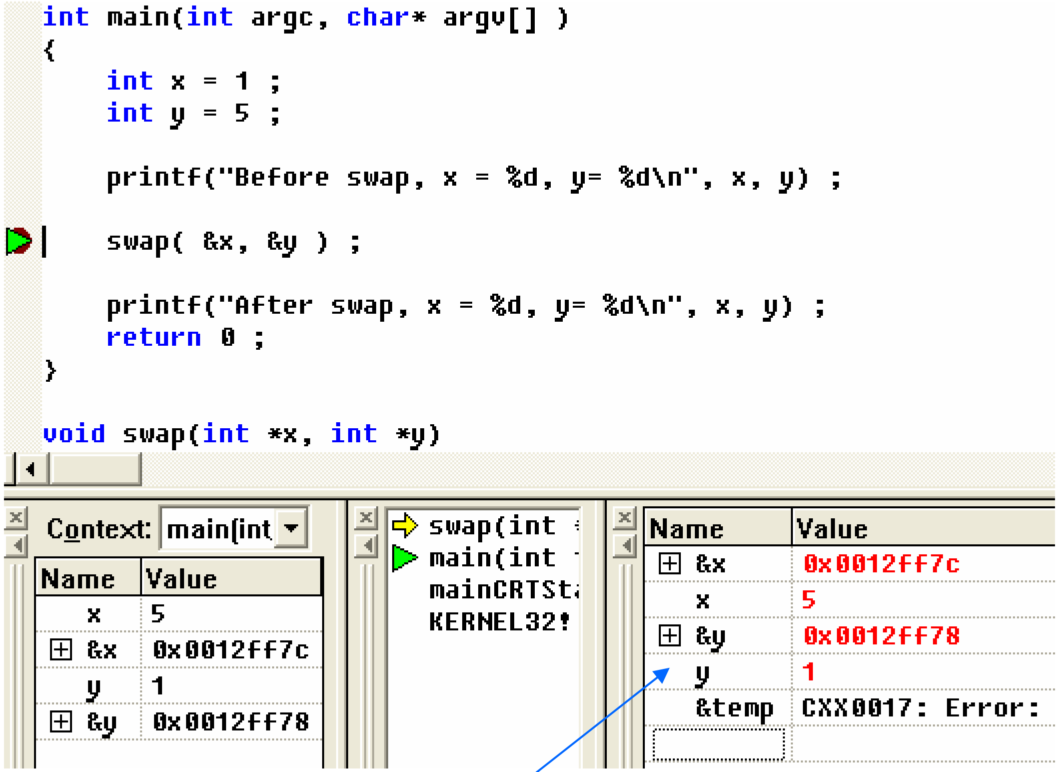
Name	Value
⊞ &x	0x0012ff24 "|ÿ■"
⊟ x	0x0012ff7c
	5
⊞ &y	0x0012ff28 "xÿ■"
⊟ y	0x0012ff78
	1
⊟ &temp	0x0012ff18
	1



address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	• • •	
0x0012FF78	1	y of main
0x0012FF7C	5	x of main

move value of temp of swap to y of main

Call graph trace [6]



```
int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

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

    swap( &x, &y ) ;

    printf("After swap, x = %d, y= %d\n", x, y) ;
    return 0 ;
}

void swap(int *x, int *y)
```

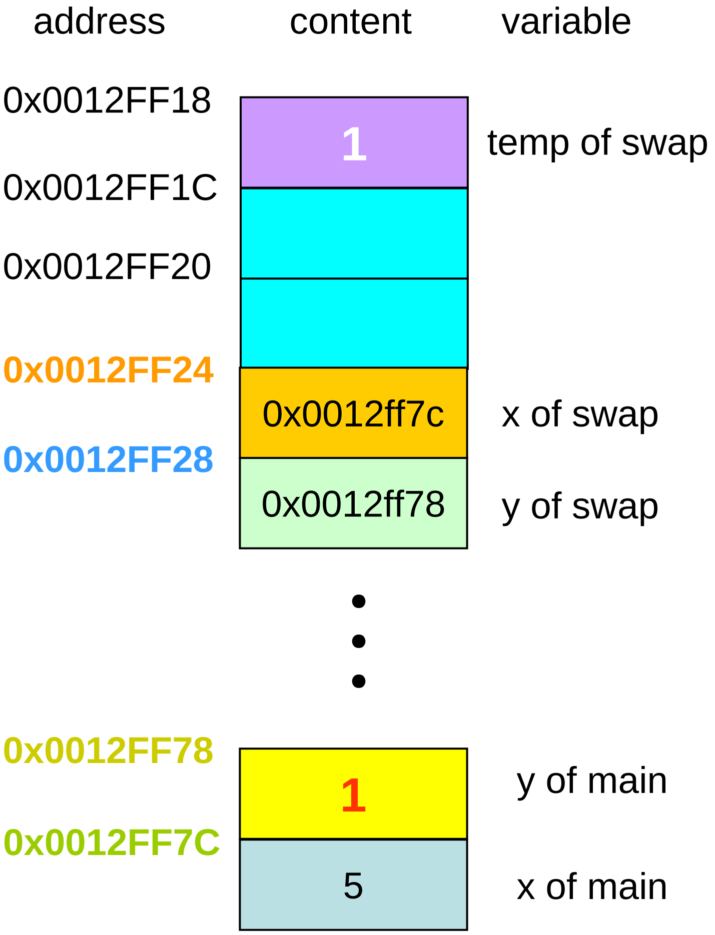
Context: main(int

Name	Value
x	5
⊞ &x	0x0012ff7c
y	1
⊞ &y	0x0012ff78

```
swap(int
main(int
mainCRTSt
KERNEL32!
```

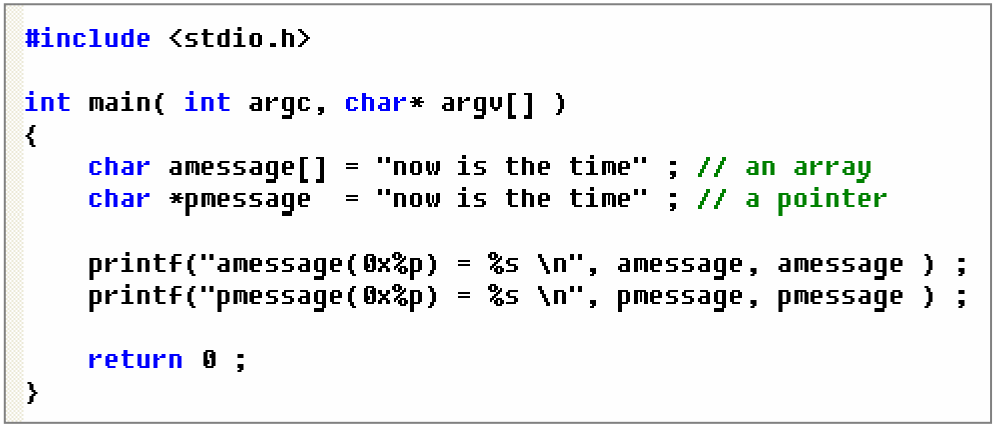
Name	Value
⊞ &x	0x0012ff7c
x	5
⊞ &y	0x0012ff78
y	1
&temp	CXX0017: Error:

y of main, check its address



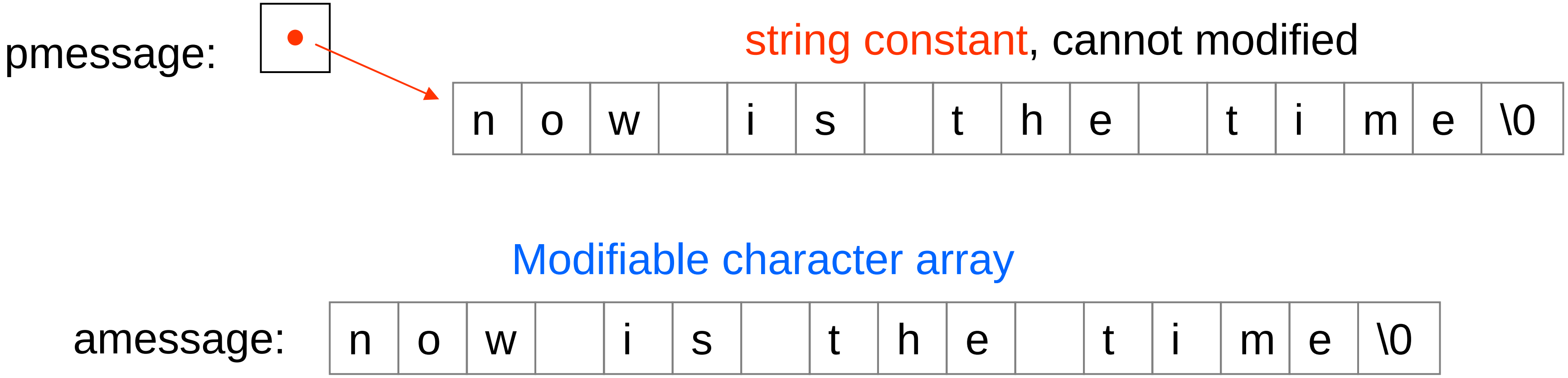
address	content	variable
0x0012FF18	1	temp of swap
0x0012FF1C		
0x0012FF20		
0x0012FF24	0x0012ff7c	x of swap
0x0012FF28	0x0012ff78	y of swap
	• • •	
0x0012FF78	1	y of main
0x0012FF7C	5	x of main

Character array v.s. character pointer [1]



```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    char amessage[] = "now is the time" ; // an array
    char *pmessage  = "now is the time" ; // a pointer

    printf("amessage(0x%p) = %s \n", amessage, amessage ) ;
    printf("pmessage(0x%p) = %s \n", pmessage, pmessage ) ;

    return 0 ;
}
```

Compiler determine length of string and then size of amessage



pmessage: ●

string constant, cannot modified

n	o	w		i	s		t	h	e		t	i	m	e	\0

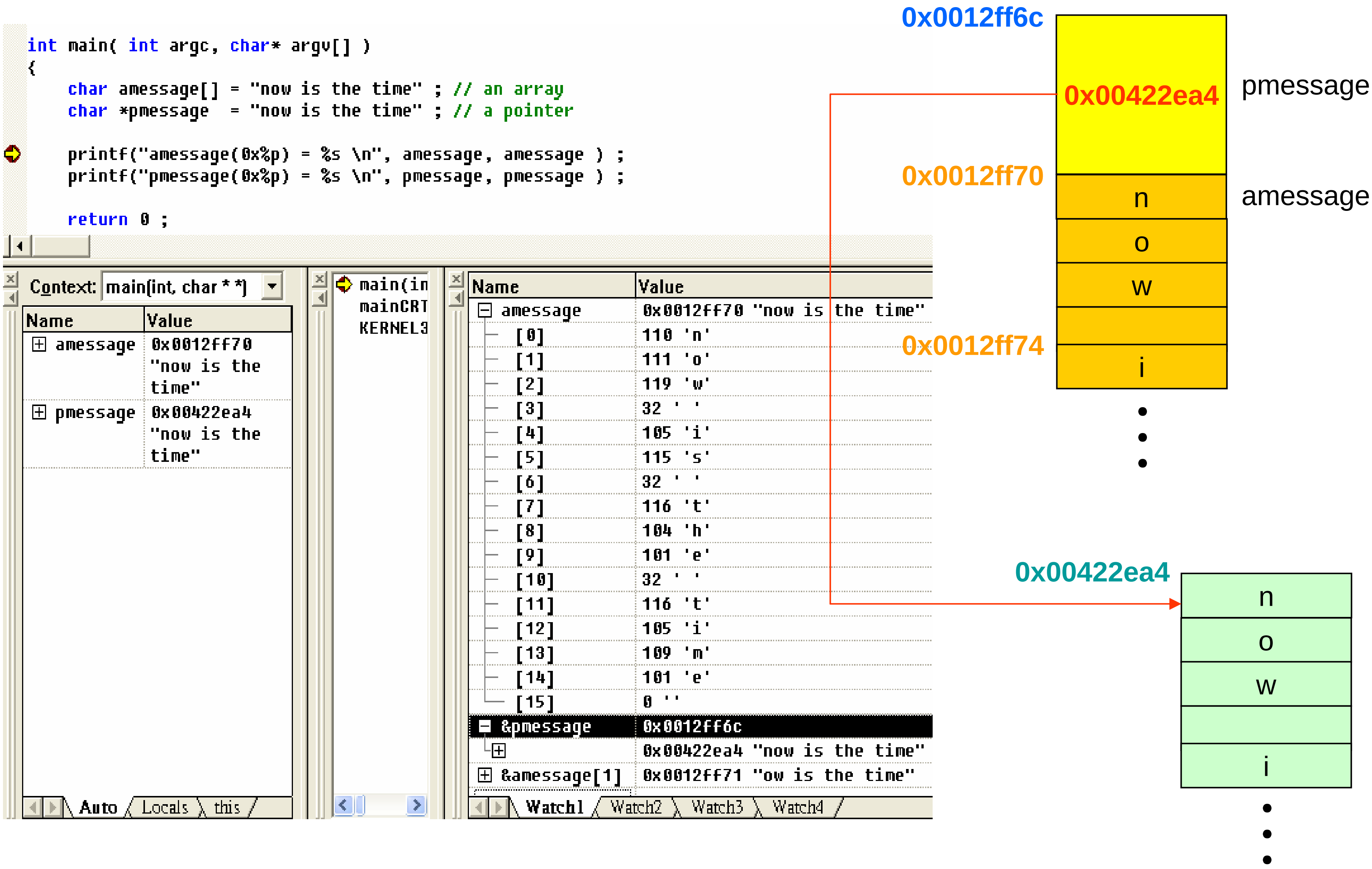
Modifiable character array

amessage:

n	o	w		i	s		t	h	e		t	i	m	e	\0

*pmessage is a pointer, initialized to point to a string constant; the pointer may subsequently be modified to point elsewhere, but the result is undefined uf you try to modify the string contents

Character array v.s. character pointer [2]



```
int main( int argc, char* argv[] )
{
    char amessage[] = "now is the time" ; // an array
    char *pmessage  = "now is the time" ; // a pointer

    printf("amessage(0x%p) = %s \n", amessage, amessage ) ;
    printf("pmessage(0x%p) = %s \n", pmessage, pmessage ) ;

    return 0 ;
```

Context: main(int, char * *)

Name	Value
⊞ amessage	0x0012ff70 "now is the time"
⊞ pmessage	0x00422ea4 "now is the time"

Auto / Locals / this

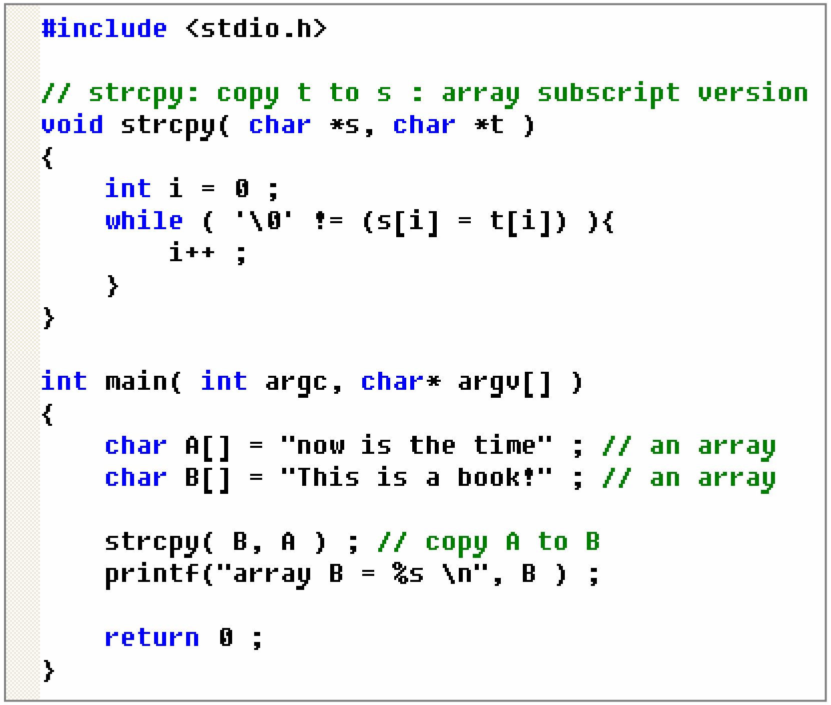
main(in
mainCRT
KERNEL3

Name	Value
⊟ amessage	0x0012ff70 "now is the time"
[0]	110 'n'
[1]	111 'o'
[2]	119 'w'
[3]	32 ' '
[4]	105 'i'
[5]	115 's'
[6]	32 ' '
[7]	116 't'
[8]	104 'h'
[9]	101 'e'
[10]	32 ' '
[11]	116 't'
[12]	105 'i'
[13]	109 'm'
[14]	101 'e'
[15]	0 ''
⊟ &pmessage	0x0012ff6c
⊞	0x00422ea4 "now is the time"
⊞ &amessage[1]	0x0012ff71 "ow is the time"

Watch1 / Watch2 / Watch3 / Watch4

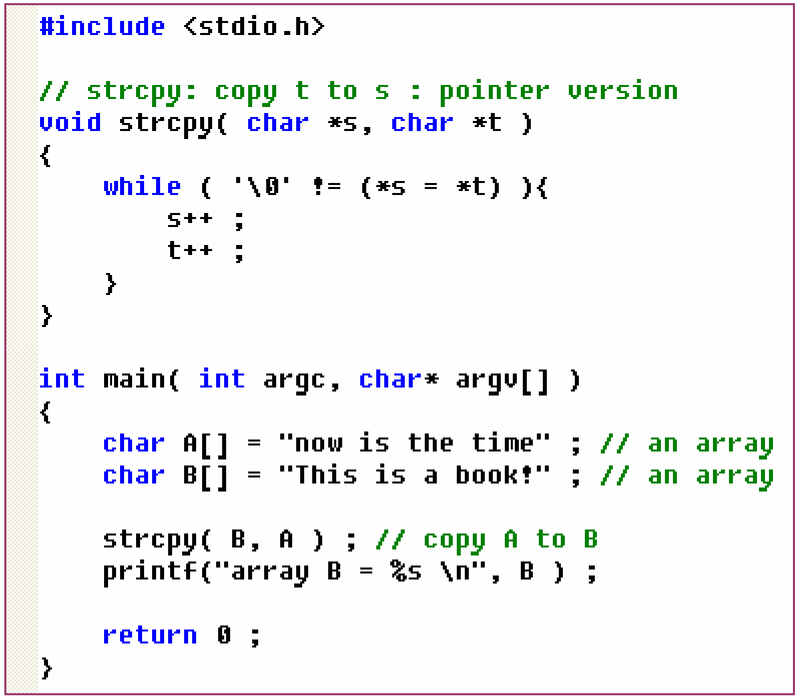
Fixed pointer v.s. movable pointer

s, ***t*** are regarded as array name, fixed, use index i to sweep entire character string



```
#include <stdio.h>

// strcpy: copy t to s : array subscript version
void strcpy( char *s, char *t )
{
    int i = 0 ;
    while ( '\0' != (s[i] = t[i]) ){
        i++ ;
    }
}

int main( int argc, char* argv[] )
{
    char A[] = "now is the time" ; // an array
    char B[] = "This is a book!" ; // an array

    strcpy( B, A ) ; // copy A to B
    printf("array B = %s \n", B ) ;

    return 0 ;
}
```

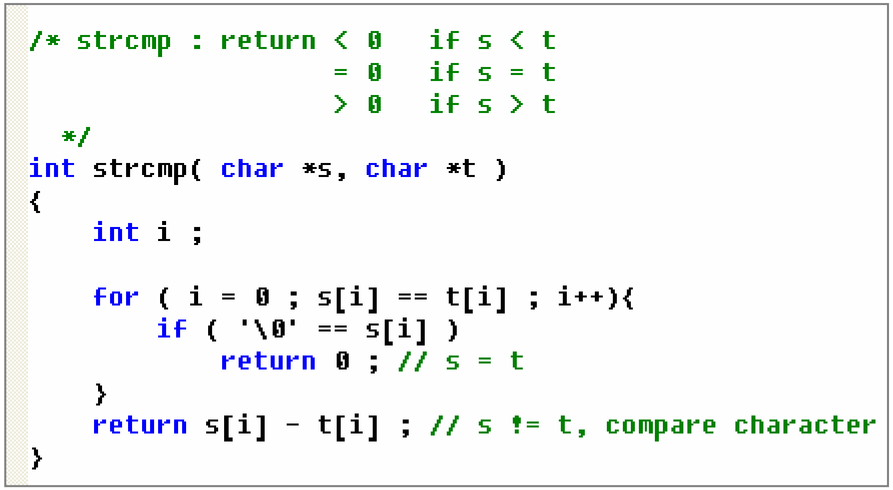
s, ***t*** are pointers, movable, s and t sweep entire character string



```
#include <stdio.h>

// strcpy: copy t to s : pointer version
void strcpy( char *s, char *t )
{
    while ( '\0' != (*s = *t) ){
        s++ ;
        t++ ;
    }
}

int main( int argc, char* argv[] )
{
    char A[] = "now is the time" ; // an array
    char B[] = "This is a book!" ; // an array

    strcpy( B, A ) ; // copy A to B
    printf("array B = %s \n", B ) ;

    return 0 ;
}
```

Question: when ***s*** and ***t*** move in function ***strcpy***, why array A and B in function main are fixed?

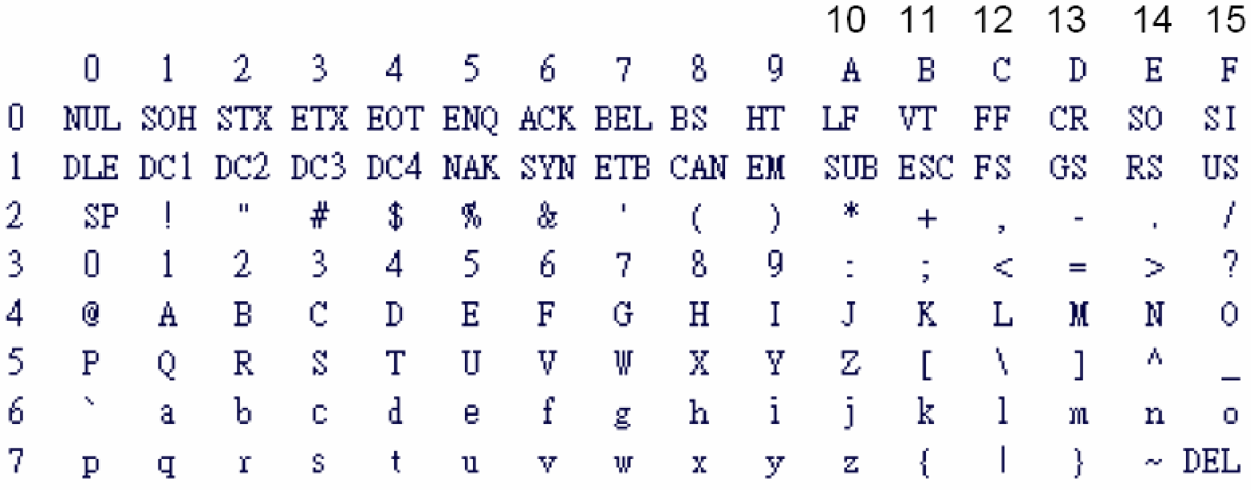
String comparison



```
/* strcmp : return < 0   if s < t
                   = 0   if s = t
                   > 0   if s > t
  */
int strcmp( char *s, char *t )
{
    int i ;

    for ( i = 0 ; s[i] == t[i] ; i++){
        if ( '\0' == s[i] )
            return 0 ; // s = t
    }
    return s[i] - t[i] ; // s != t, compare character
}
```

Lexicographic order of string ***s*** and ***t***

We say ***s < t*** if there exists index ***k*** such that ***s[k] < t[k]*** in ASCII code sense.



											10	11	12	13	14	15
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

A-Z occupies 0x41 ~ 0x5A

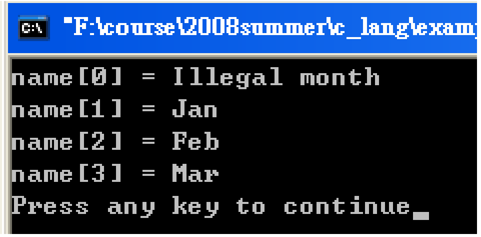
a-z occupies 0x61 ~ 0x7A

OutLine

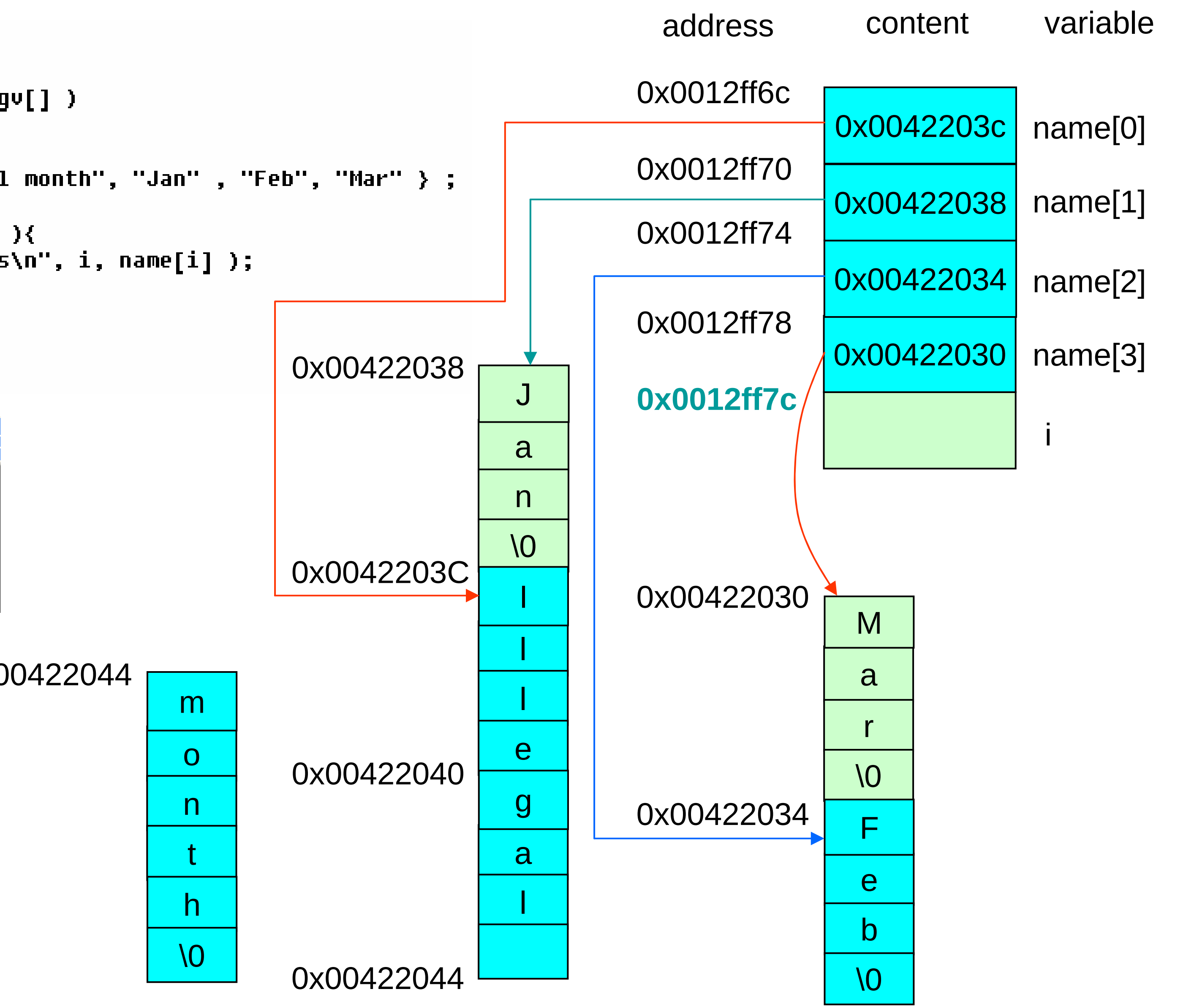
- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

Pointer array: pointers to pointers [1]

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i ;
    char *name[] = { "Illegal month", "Jan" , "Feb", "Mar" } ;

    for( i = 0 ; i < 4 ; i++ ){
        printf("name[%d] = %s\n", i, name[i] );
    }

    return 0 ;
}
```

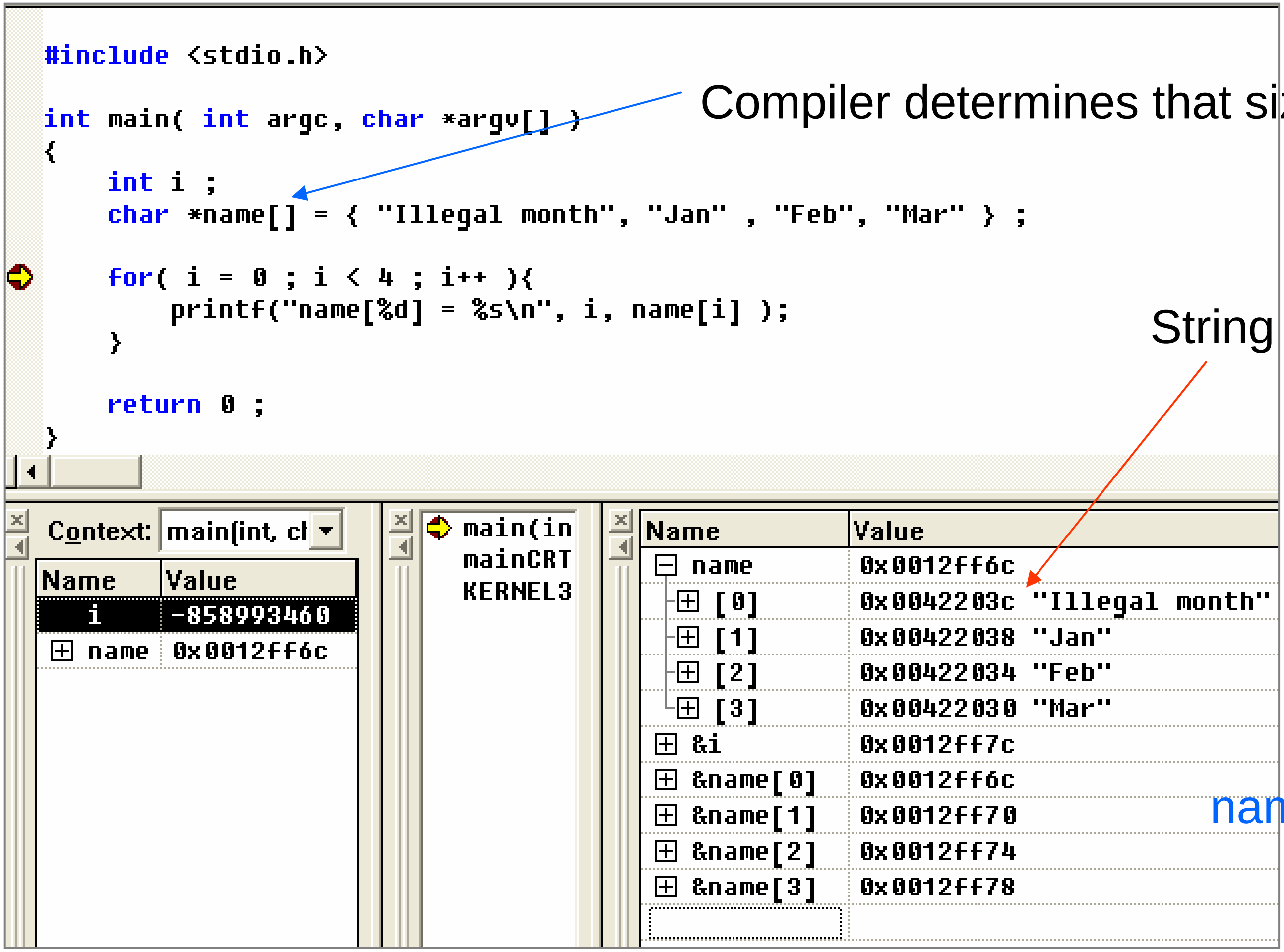


```
name[0] = Illegal month
name[1] = Jan
name[2] = Feb
name[3] = Mar
Press any key to continue_
```



address	content	variable
0x0012ff6c	0x0042203c	name[0]
0x0012ff70	0x00422038	name[1]
0x0012ff74	0x00422034	name[2]
0x0012ff78	0x00422030	name[3]
0x0012ff7c		i

Pointer array: pointers to pointers [2]



```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i ;
    char *name[] = { "Illegal month", "Jan" , "Feb", "Mar" } ;

    for( i = 0 ; i < 4 ; i++ ){
        printf("name[%d] = %s\n", i, name[i] );
    }

    return 0 ;
}
```

Compiler determines that size of array is 4

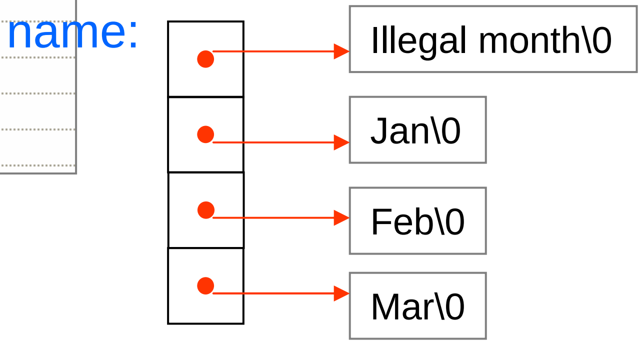
String starts at 0x0042203c

Name	Value
i	-858993460
⊞ name	0x0012ff6c

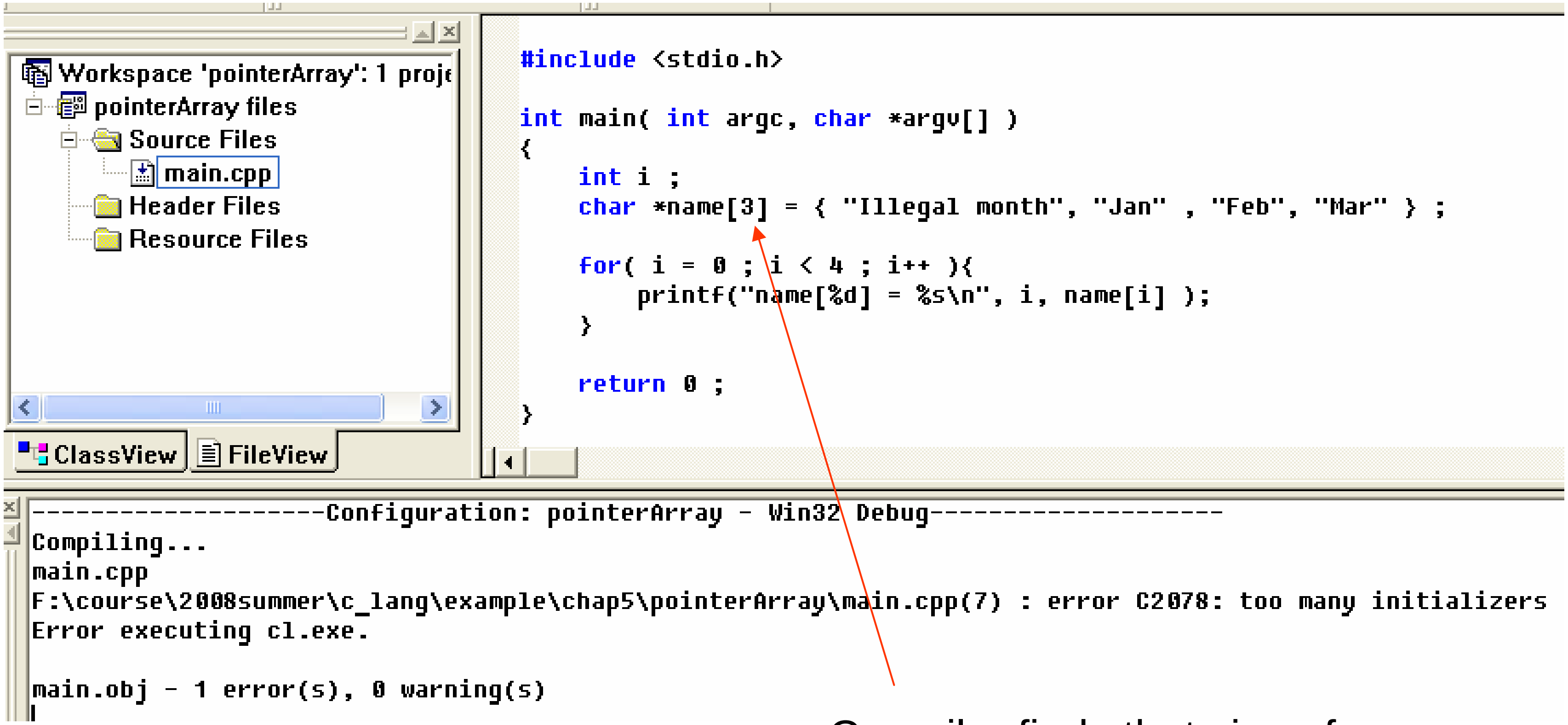
```
main(in
mainCRT
KERNEL3
```

Name	Value
⊟ name	0x0012ff6c
⊞ [0]	0x0042203c "Illegal month"
⊞ [1]	0x00422038 "Jan"
⊞ [2]	0x00422034 "Feb"
⊞ [3]	0x00422030 "Mar"
⊞ &i	0x0012ff7c
⊞ &name[0]	0x0012ff6c
⊞ &name[1]	0x0012ff70
⊞ &name[2]	0x0012ff74
⊞ &name[3]	0x0012ff78

Symbolic representaiton



Pointer array: pointers to pointers [3]

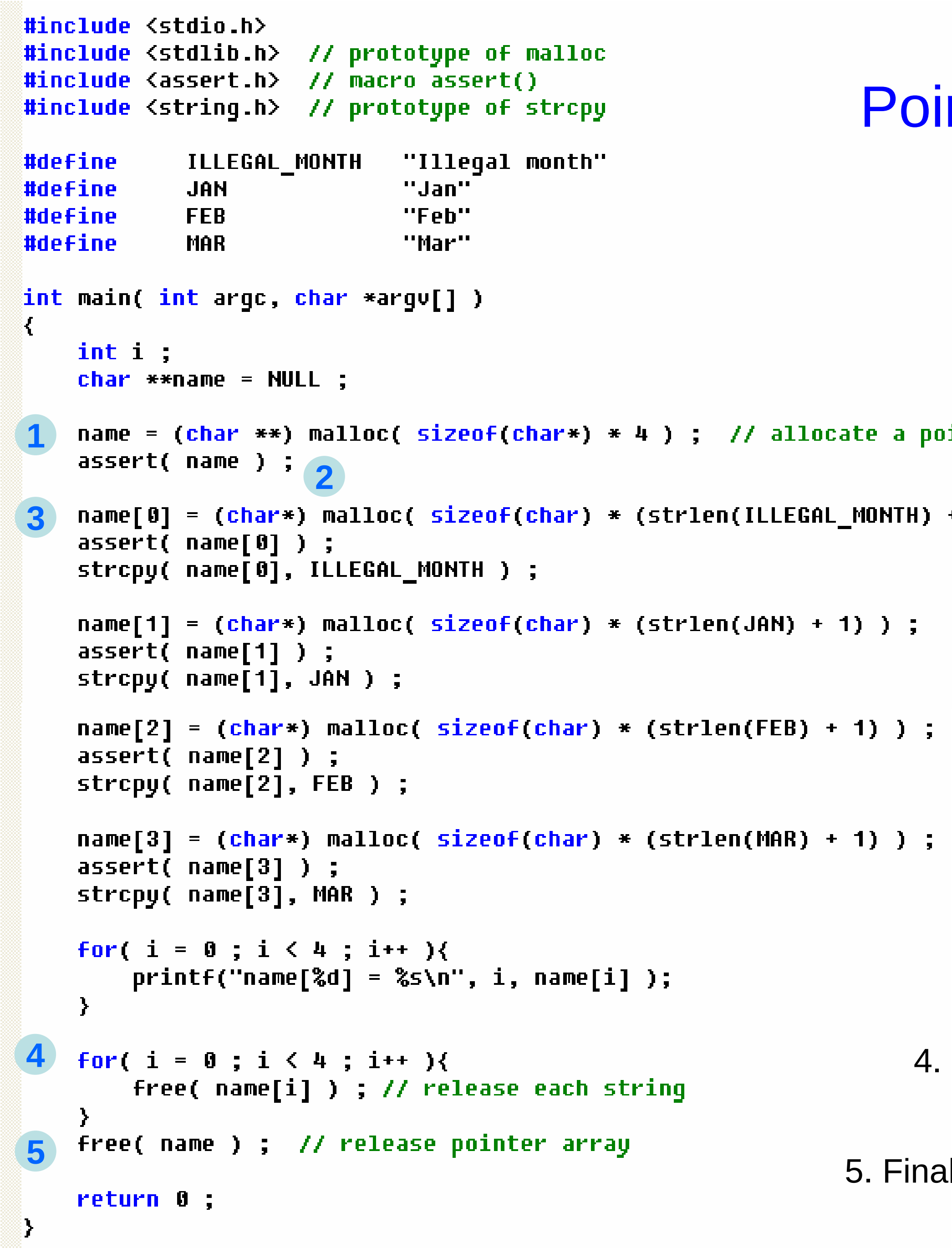


```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i ;
    char *name[3] = { "Illegal month", "Jan" , "Feb", "Mar" } ;

    for( i = 0 ; i < 4 ; i++ ){
        printf("name[%d] = %s\n", i, name[i] );
    }

    return 0 ;
}
```

```
--------------------Configuration: pointerArray - Win32 Debug--------------------
Compiling...
main.cpp
F:\course\2008summer\c_lang\example\chap5\pointerArray\main.cpp(7) : error C2078: too many initializers
Error executing cl.exe.

main.obj - 1 error(s), 0 warning(s)
```

Compiler finds that size of array should be 4 at least, this is conflict to number 3 defined by programmer

Pointer array: pointers to pointers [4]



```
#include <stdio.h>
#include <stdlib.h>  // prototype of malloc
#include <assert.h>  // macro assert()
#include <string.h>  // prototype of strcpy

#define      ILLEGAL_MONTH   "Illegal month"
#define      JAN             "Jan"
#define      FEB             "Feb"
#define      MAR             "Mar"

int main( int argc, char *argv[] )
{
    int i ;
    char **name = NULL ;

    name = (char **) malloc( sizeof(char*) * 4 ) ;  // allocate a pointer array
    assert( name ) ;

    name[0] = (char*) malloc( sizeof(char) * (strlen(ILLEGAL_MONTH) + 1) ) ;
    assert( name[0] ) ;
    strcpy( name[0], ILLEGAL_MONTH ) ;

    name[1] = (char*) malloc( sizeof(char) * (strlen(JAN) + 1) ) ;
    assert( name[1] ) ;
    strcpy( name[1], JAN ) ;

    name[2] = (char*) malloc( sizeof(char) * (strlen(FEB) + 1) ) ;
    assert( name[2] ) ;
    strcpy( name[2], FEB ) ;

    name[3] = (char*) malloc( sizeof(char) * (strlen(MAR) + 1) ) ;
    assert( name[3] ) ;
    strcpy( name[3], MAR ) ;

    for( i = 0 ; i < 4 ; i++ ){
        printf("name[%d] = %s\n", i, name[i] );
    }

    for( i = 0 ; i < 4 ; i++ ){
        free( name[i] ) ; // release each string
    }
    free( name ) ;  // release pointer array

    return 0 ;
}
```

1. allocate pointer array of 4 elements

2. Macro assert is used as diagnosis, see page 253 in textbook

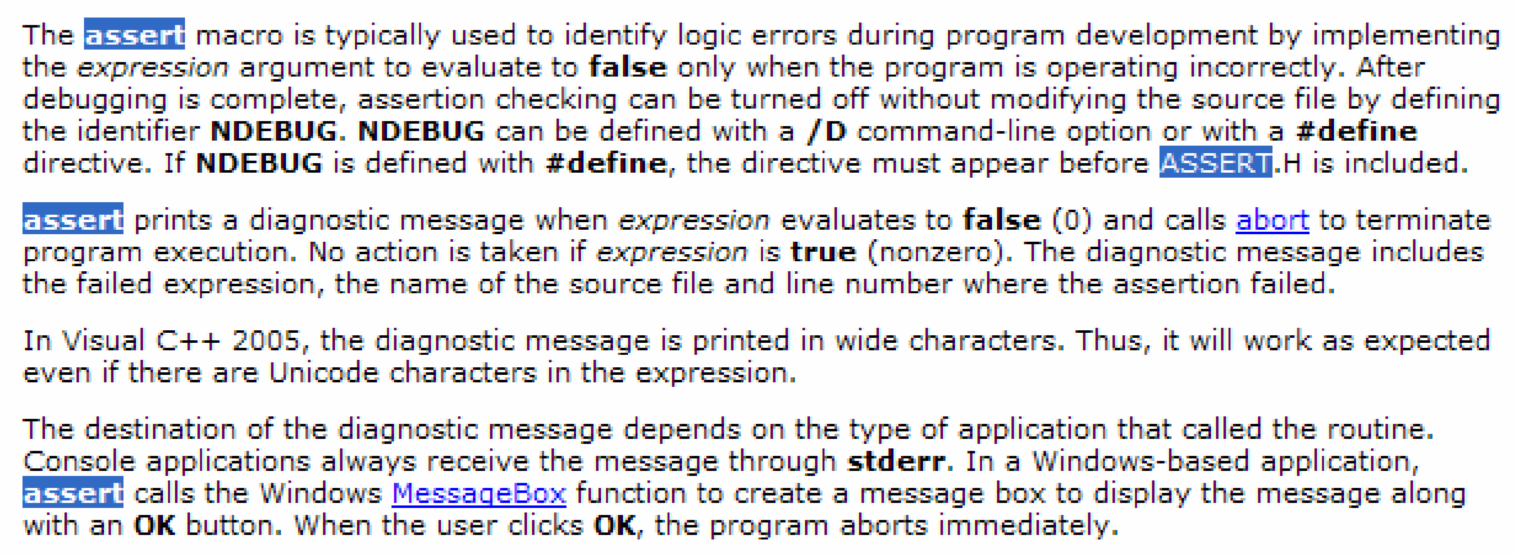
3. allocate character array, note that we need one more space for \0

4. First, release each string

5. Finally release pointer array

Diagnosis Macro: assert

void assert(int expression)

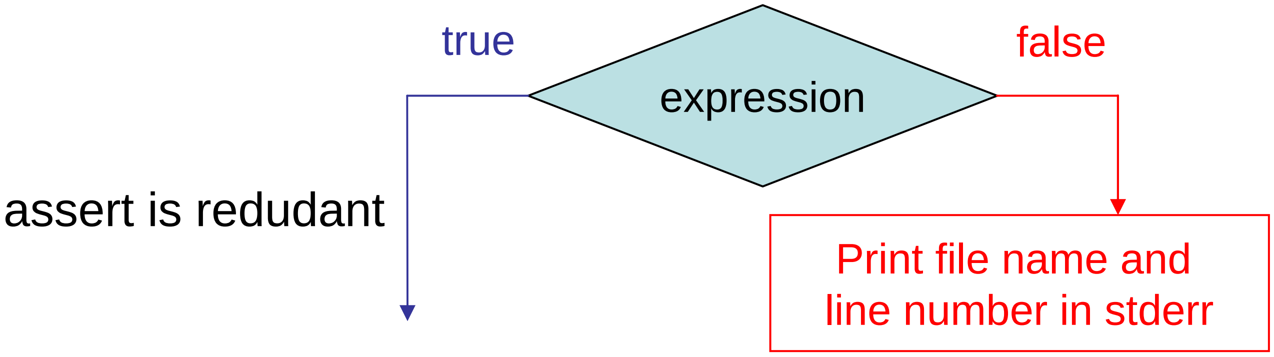


The assert macro is typically used to identify logic errors during program development by implementing the *expression* argument to evaluate to **false** only when the program is operating incorrectly. After debugging is complete, assertion checking can be turned off without modifying the source file by defining the identifier **NDEBUG**. **NDEBUG** can be defined with a **/D** command-line option or with a **#define** directive. If **NDEBUG** is defined with **#define**, the directive must appear before ASSERT.H is included.

assert prints a diagnostic message when *expression* evaluates to **false** (0) and calls abort to terminate program execution. No action is taken if *expression* is **true** (nonzero). The diagnostic message includes the failed expression, the name of the source file and line number where the assertion failed.

In Visual C++ 2005, the diagnostic message is printed in wide characters. Thus, it will work as expected even if there are Unicode characters in the expression.

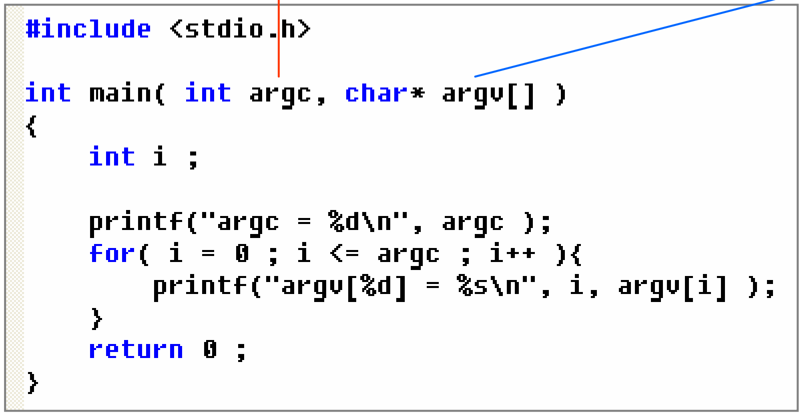
The destination of the diagnostic message depends on the type of application that called the routine. Console applications always receive the message through **stderr**. In a Windows-based application, assert calls the Windows MessageBox function to create a message box to display the message along with an **OK** button. When the user clicks **OK**, the program aborts immediately.



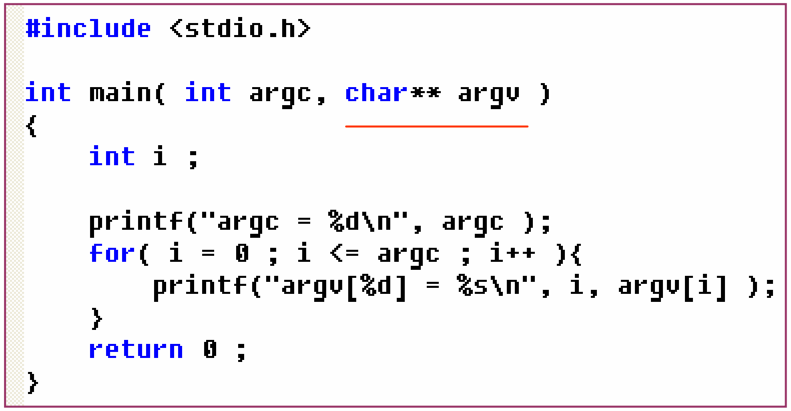
Command-line arguments   [1]

argc: argument count

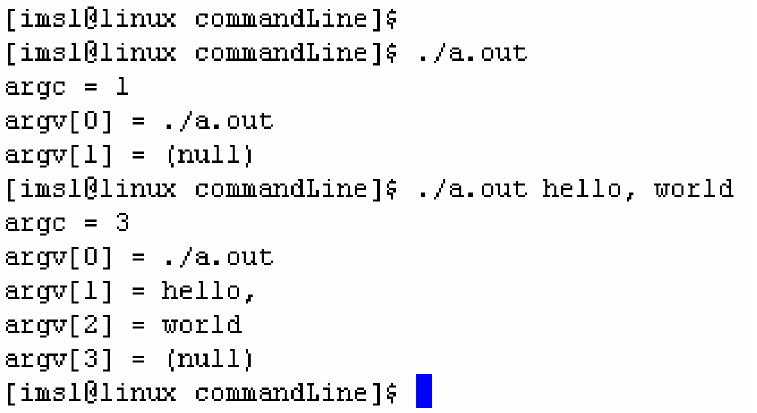
argv: argument vector



```c
#include <stdio.h>

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

    printf("argc = %d\n", argc );
    for( i = 0 ; i <= argc ; i++ ){
        printf("argv[%d] = %s\n", i, argv[i] );
    }
    return 0 ;
}
```

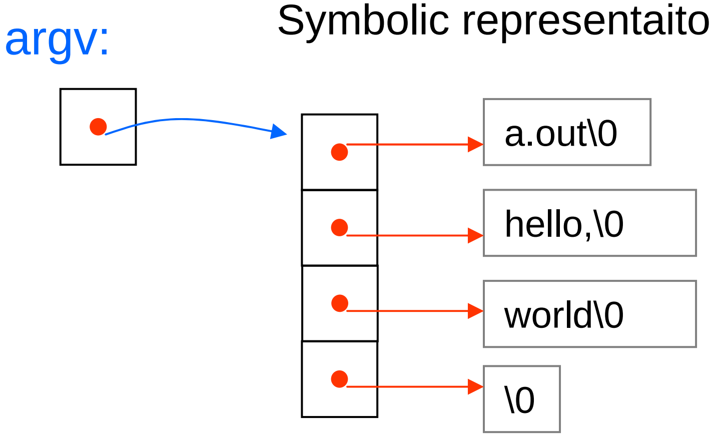


```c
#include <stdio.h>

int main( int argc, char** argv )
{
    int i ;

    printf("argc = %d\n", argc );
    for( i = 0 ; i <= argc ; i++ ){
        printf("argv[%d] = %s\n", i, argv[i] );
    }
    return 0 ;
}
```

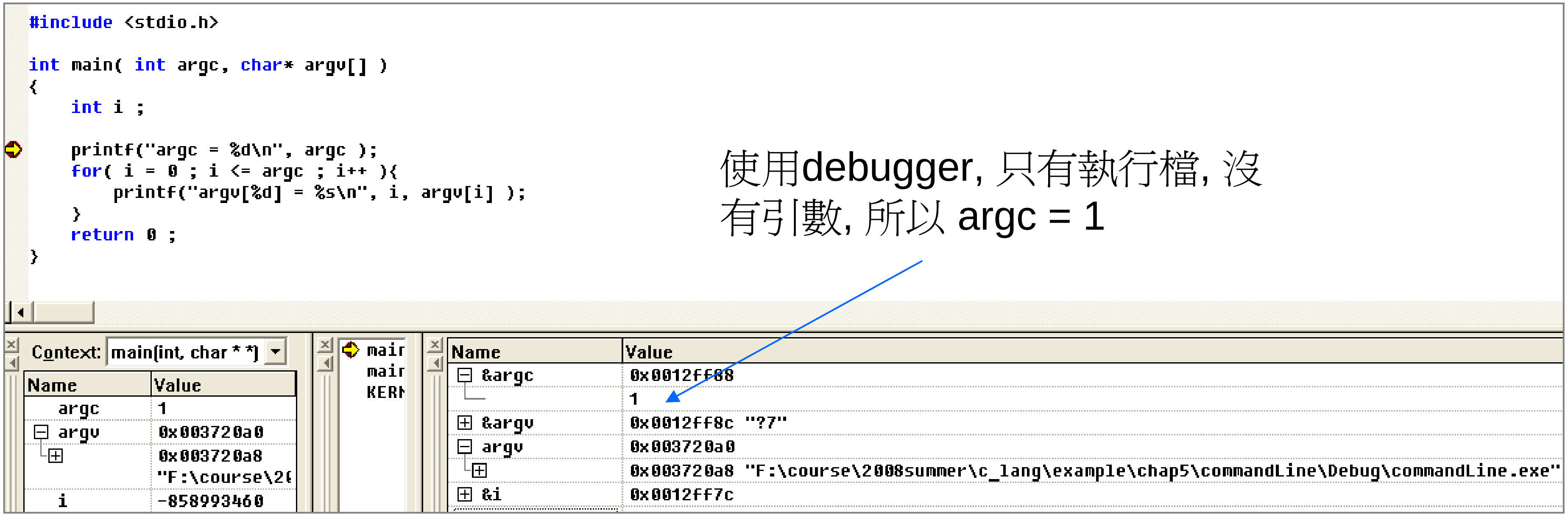


```
[imsl@linux commandLine]$
[imsl@linux commandLine]$ ./a.out
argc = 1
argv[0] = ./a.out
argv[1] = (null)
[imsl@linux commandLine]$ ./a.out hello, world
argc = 3
argv[0] = ./a.out
argv[1] = hello,
argv[2] = world
argv[3] = (null)
[imsl@linux commandLine]$ 
```

Symbolic representaiton



Command-line arguments [2]



```
#include <stdio.h>

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

    printf("argc = %d\n", argc );
    for( i = 0 ; i <= argc ; i++ ){
        printf("argv[%d] = %s\n", i, argv[i] );
    }
    return 0 ;
}
```

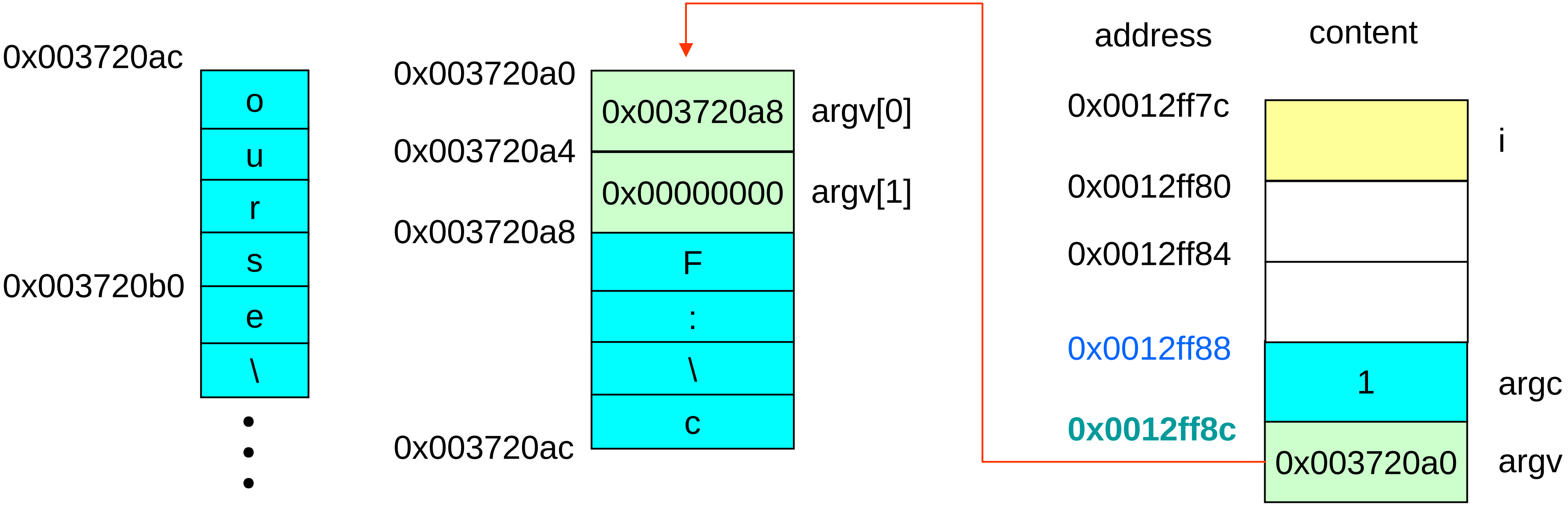
Context: main(int, char * *)

Name	Value
argc	1
argv	0x003720a0
	0x003720a8 "F:\course\2(
i	-858993460

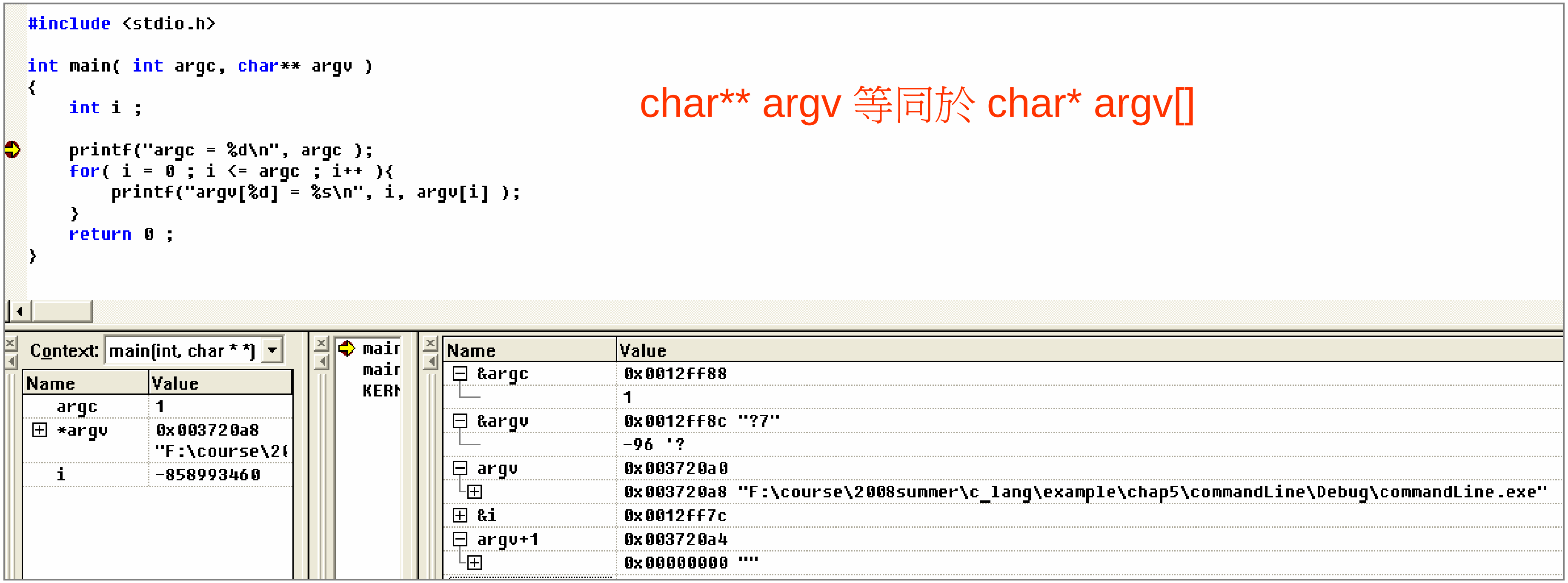
```
mair
mair
KERI
```

Name	Value
&argc	0x0012ff88
	1
&argv	0x0012ff8c "?7"
argv	0x003720a0
	0x003720a8 "F:\course\2008summer\c_lang\example\chap5\commandLine\Debug\commandLine.exe"
&i	0x0012ff7c

使用debugger, 只有執行檔, 沒有引數, 所以 argc = 1



Command-line arguments [3]



```c
#include <stdio.h>

int main( int argc, char** argv )
{
    int i ;

    printf("argc = %d\n", argc );
    for( i = 0 ; i <= argc ; i++ ){
        printf("argv[%d] = %s\n", i, argv[i] );
    }
    return 0 ;
}
```

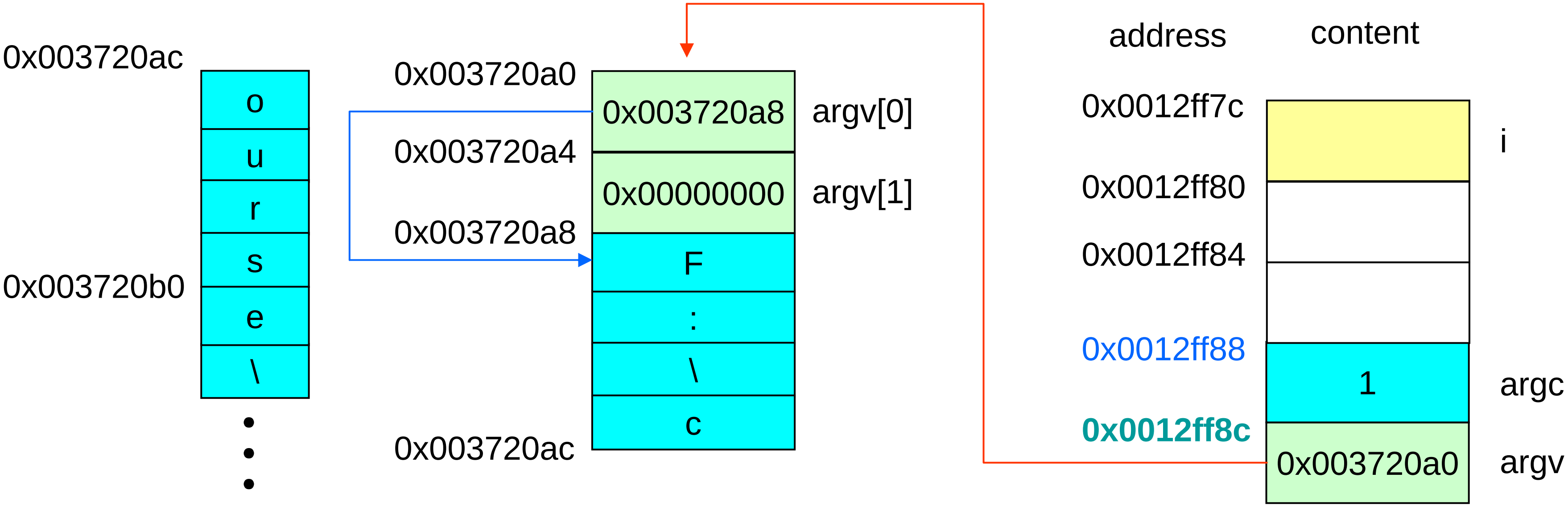
char** argv 等同於 char* argv[]

Context: main(int, char * *)

Name	Value
argc	1
⊞ *argv	0x003720a8 "F:\course\2(
i	-858993460

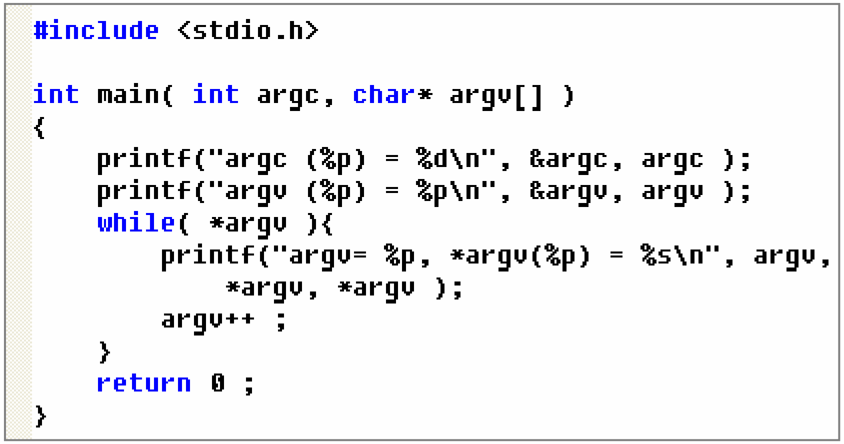
main
main
KERN

Name	Value
⊟ &argc	0x0012ff88
	1
⊟ &argv	0x0012ff8c "?7"
	-96 '?
⊟ argv	0x003720a0
⊞	0x003720a8 "F:\course\2008summer\c_lang\example\chap5\commandLine\Debug\commandLine.exe"
⊞ &i	0x0012ff7c
⊟ argv+1	0x003720a4
⊞	0x00000000 ""

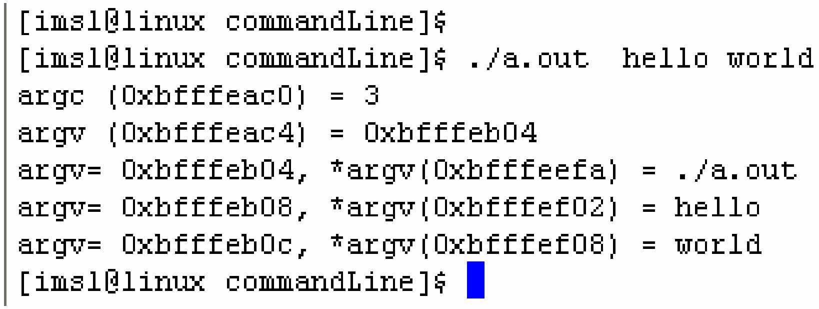


address	content	
0x0012ff7c		i
0x0012ff80		
0x0012ff84		
0x0012ff88	1	argc
0x0012ff8c	0x003720a0	argv

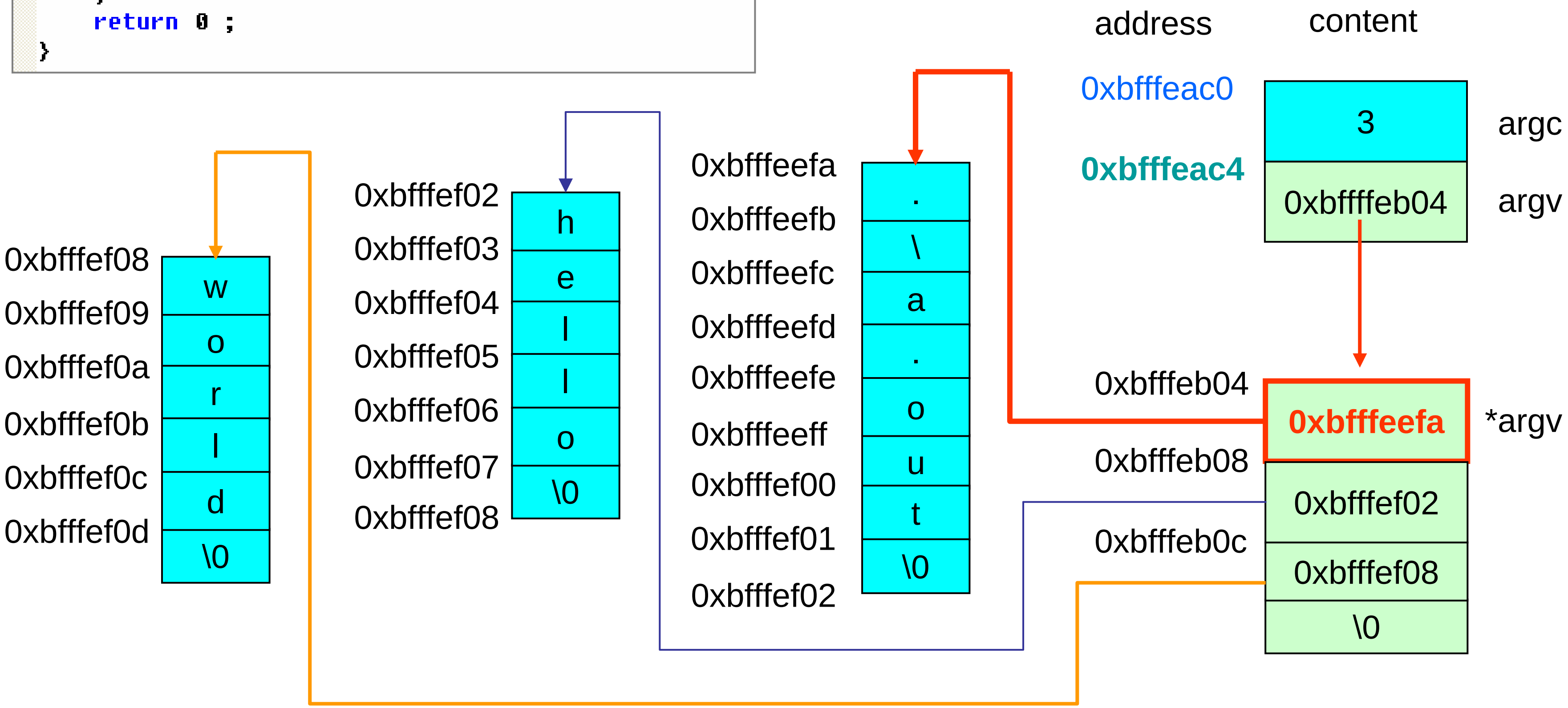
Command-line arguments [4]



```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    printf("argc (%p) = %d\n", &argc, argc );
    printf("argv (%p) = %p\n", &argv, argv );
    while( *argv ){
        printf("argv= %p, *argv(%p) = %s\n", argv,
            *argv, *argv );
        argv++ ;
    }
    return 0 ;
}
```



```
[imsl@linux commandLine]$
[imsl@linux commandLine]$ ./a.out  hello world
argc (0xbfffeac0) = 3
argv (0xbfffeac4) = 0xbfffeb04
argv= 0xbfffeb04, *argv(0xbfffeefa) = ./a.out
argv= 0xbfffeb08, *argv(0xbfffef02) = hello
argv= 0xbfffeb0c, *argv(0xbfffef08) = world
[imsl@linux commandLine]$ 
```



Command-line arguments [5]

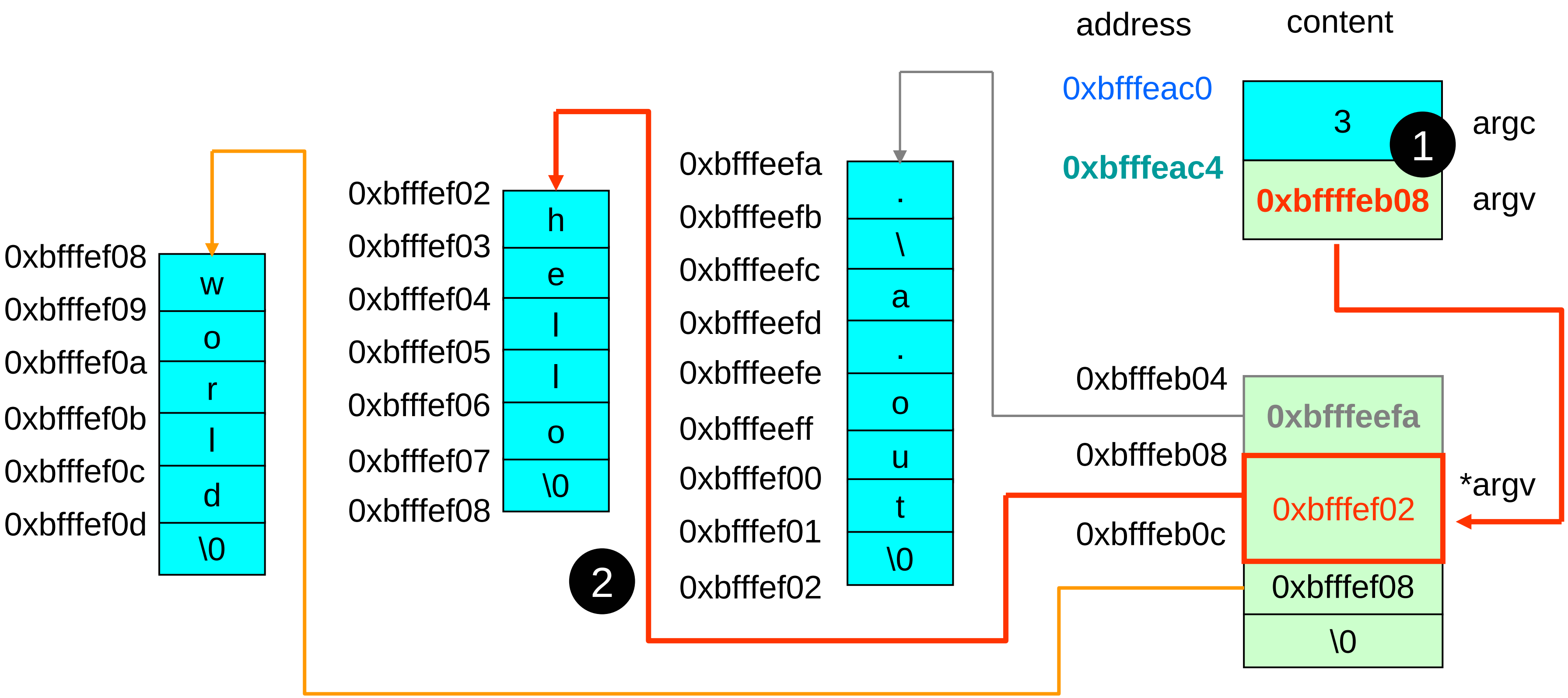
1 argv++

2 print string whose address is in *argv

char** argv

char* (*argv)

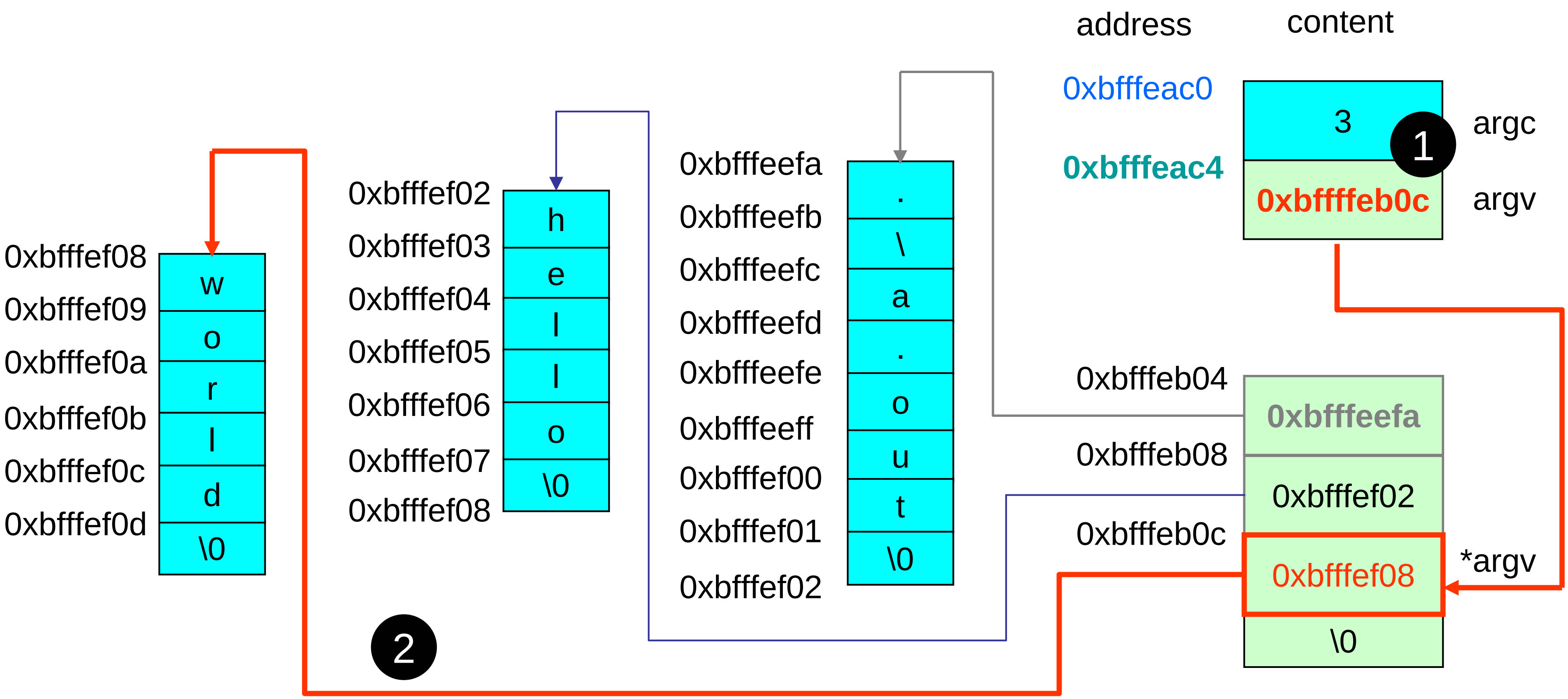
char (**argv)



Command-line arguments [6]

1 argv++

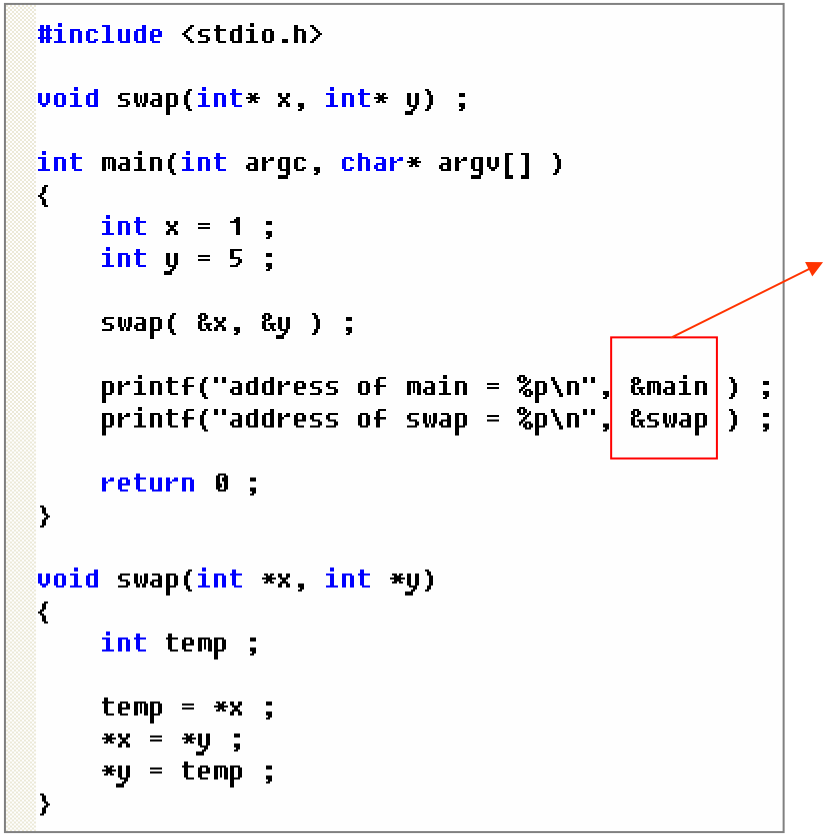
2 print string whose address is in *argv



OutLine

- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

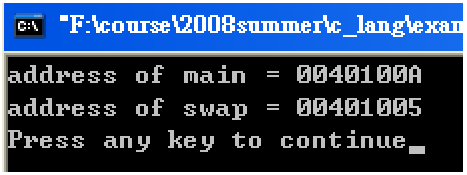
Function = address [1]



```
#include <stdio.h>

void swap(int* x, int* y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    swap( &x, &y ) ;

    printf("address of main = %p\n", &main ) ;
    printf("address of swap = %p\n", &swap ) ;

    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

Function pointer is pointer which points to address of a function

Each function has an address, we can use reference operator **&** to extract its address, however since function is equal to an address (this is different from variable), sometimes we neglect operator **&**



```
address of main = 0040100A
address of swap = 00401005
Press any key to continue_
```

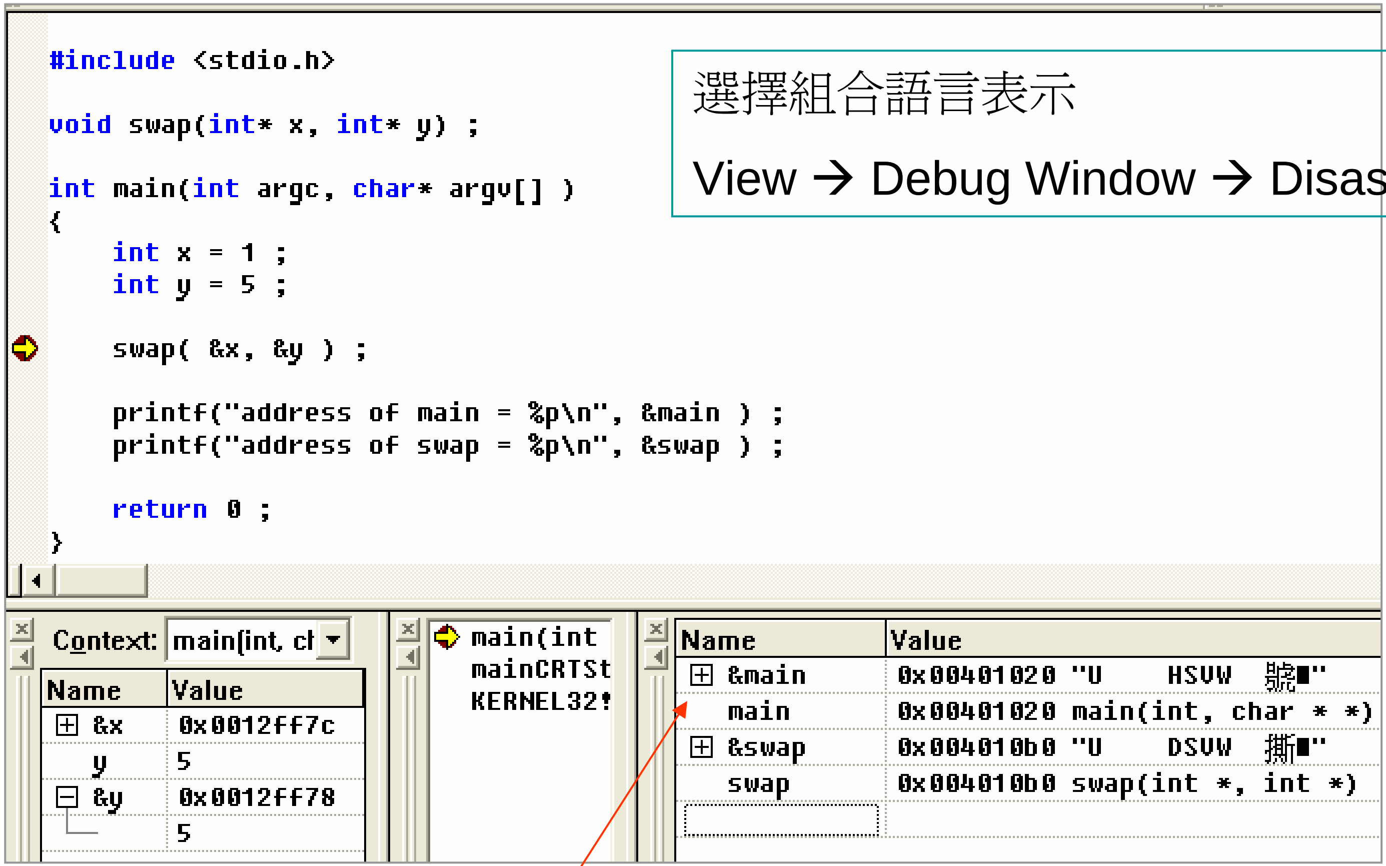
Question: a function name is equal to an address, why?

Function = address   [2]

選擇組合語言表示

View → Debug Window → Disassembly

中斷點



```
#include <stdio.h>

void swap(int* x, int* y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    swap( &x, &y ) ;

    printf("address of main = %p\n", &main ) ;
    printf("address of swap = %p\n", &swap ) ;

    return 0 ;
}
```

Context: main(int, ch

Name	Value
⊞ &x	0x0012ff7c
y	5
⊟ &y	0x0012ff78
	5

```
main(int
mainCRTSt
KERNEL32!
```

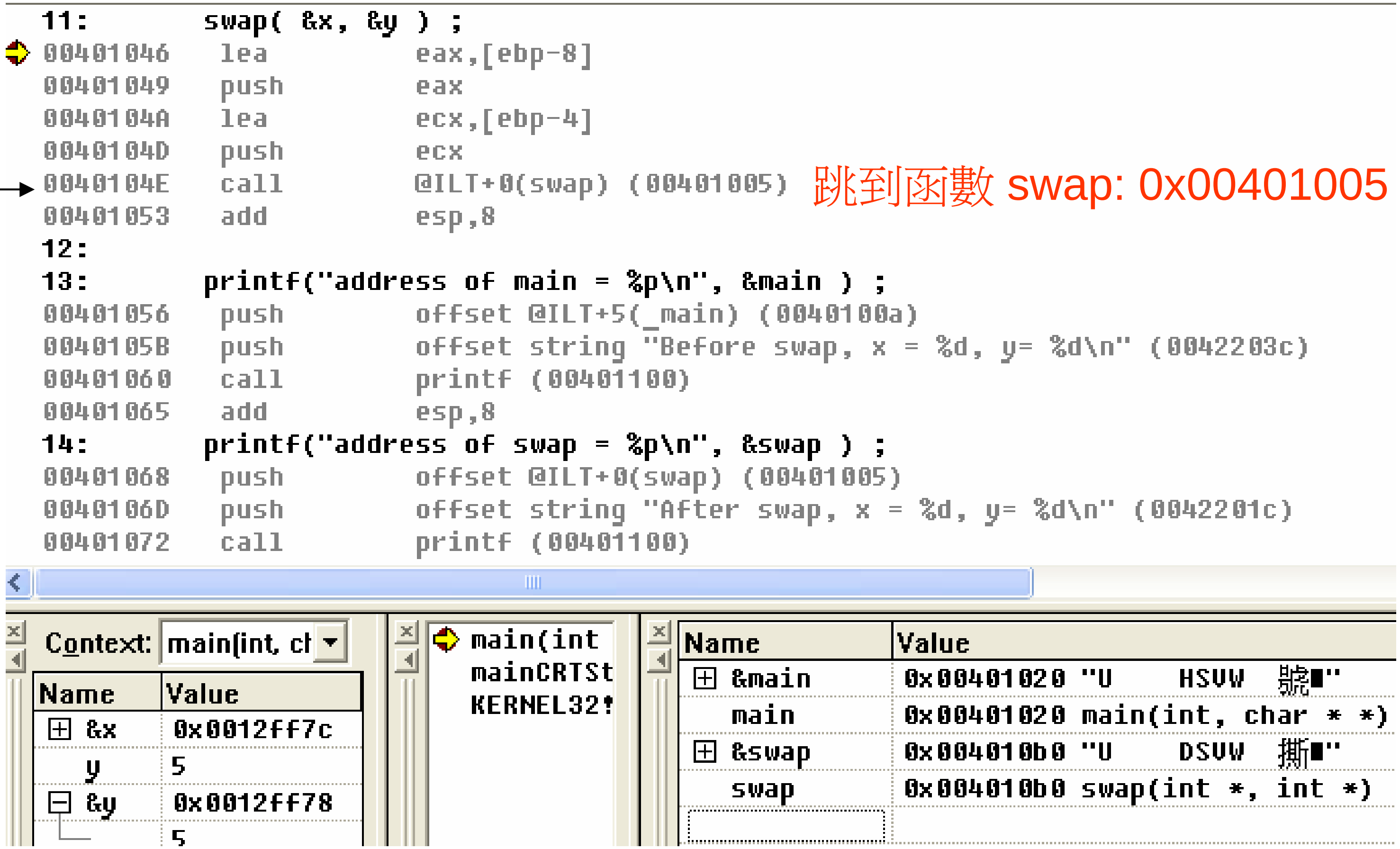
Name	Value
⊞ &main	0x00401020 "U HSVW 號■"
main	0x00401020 main(int, char * *)
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap	0x004010b0 swap(int *, int *)

main 和 &main 有相同值, 因爲函數名等價於位址

Question: why main = 0x00401020 in debugger but

main = 0x0040100A in ouput

Function = address [3]



```
11:         swap( &x, &y ) ;
00401046   lea          eax,[ebp-8]
00401049   push         eax
0040104A   lea          ecx,[ebp-4]
0040104D   push         ecx
0040104E   call         @ILT+0(swap) (00401005)
00401053   add          esp,8
12:
13:         printf("address of main = %p\n", &main ) ;
00401056   push         offset @ILT+5(_main) (0040100a)
0040105B   push         offset string "Before swap, x = %d, y= %d\n" (0042203c)
00401060   call         printf (00401100)
00401065   add          esp,8
14:         printf("address of swap = %p\n", &swap ) ;
00401068   push         offset @ILT+0(swap) (00401005)
0040106D   push         offset string "After swap, x = %d, y= %d\n" (0042201c)
00401072   call         printf (00401100)
```

跳到函數 swap: 0x00401005

Context: main(int, ch

Name	Value
⊞ &x	0x0012ff7c
y	5
⊟ &y	0x0012ff78
└	5

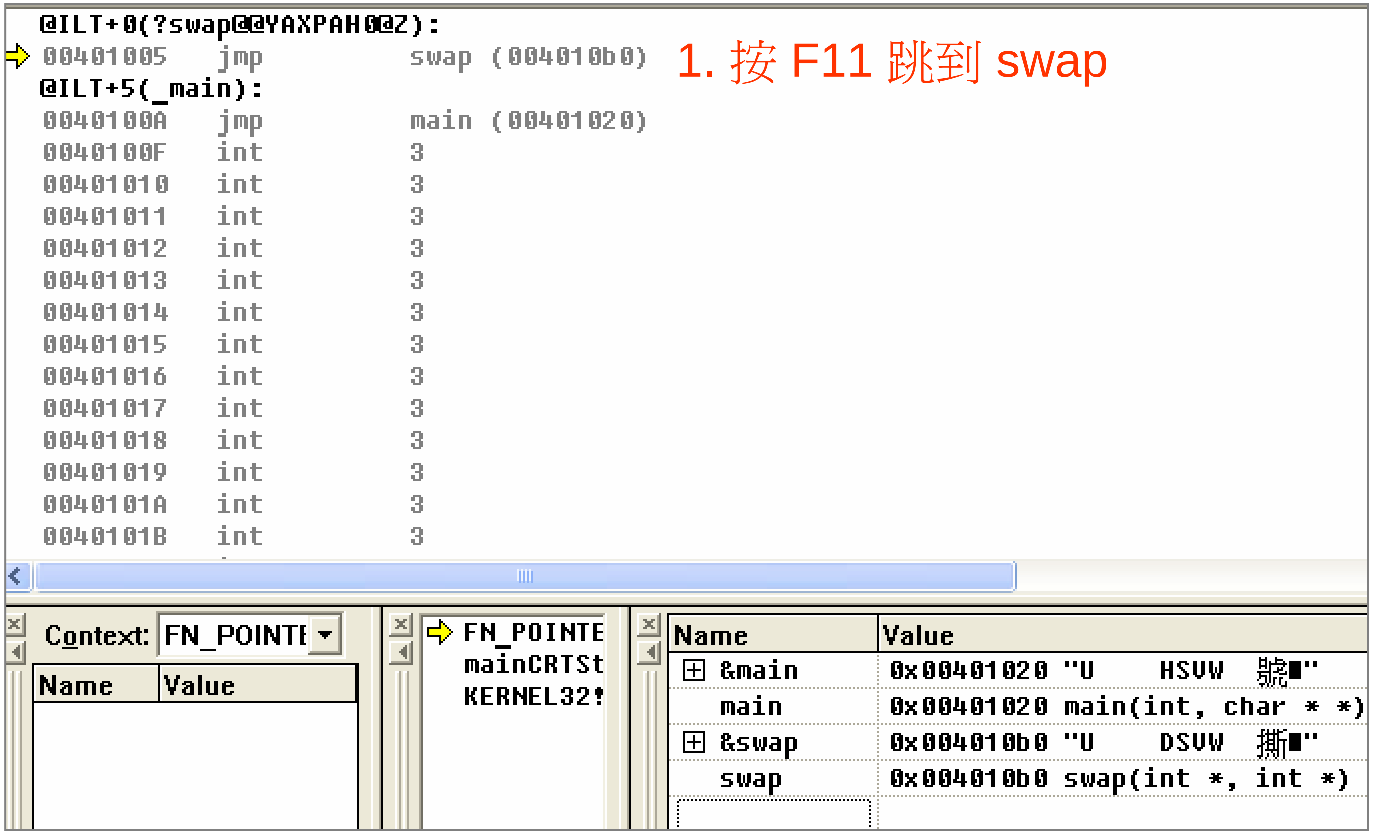
```
main(int
mainCRTSt
KERNEL32!
```

Name	Value
⊞ &main	0x00401020 "U HSVW 號■"
main	0x00401020 main(int, char * *)
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap	0x004010b0 swap(int *, int *)

按 F11進入swap

Function = address [4]

在位址0x00401005內的值表示指令 jmp swap (0x004010b0)



```
@ILT+0(?swap@@YAXPAH0@Z):
00401005   jmp         swap (004010b0)
@ILT+5(_main):
0040100A   jmp         main (00401020)
0040100F   int         3
00401010   int         3
00401011   int         3
00401012   int         3
00401013   int         3
00401014   int         3
00401015   int         3
00401016   int         3
00401017   int         3
00401018   int         3
00401019   int         3
0040101A   int         3
0040101B   int         3
```

1. 按 F11 跳到 swap

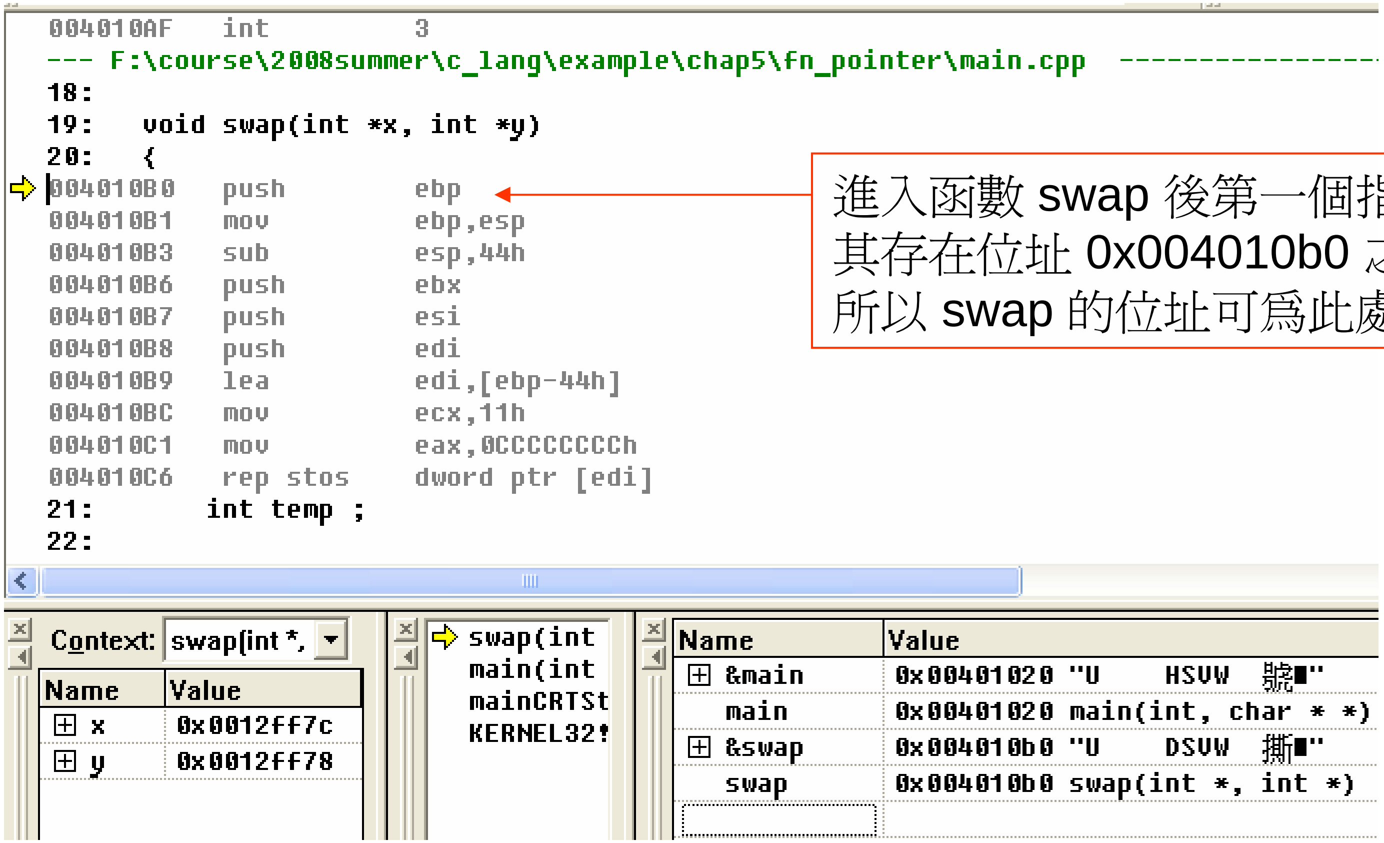
Name	Value

```
FN_POINTE
mainCRTSt
KERNEL32!
```

Name	Value
⊞ &main	0x00401020 "U HSVW 虢■"
main	0x00401020 main(int, char * *)
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap	0x004010b0 swap(int *, int *)

在位址0x0040100A內的值表示指令 jmp main (0x00401020)

Function = address [5]



```
004010AF   int          3
--- F:\course\2008summer\c_lang\example\chap5\fn_pointer\main.cpp  ----------------
18:
19:    void swap(int *x, int *y)
20:    {
004010B0   push         ebp
004010B1   mov          ebp,esp
004010B3   sub          esp,44h
004010B6   push         ebx
004010B7   push         esi
004010B8   push         edi
004010B9   lea          edi,[ebp-44h]
004010BC   mov          ecx,11h
004010C1   mov          eax,0CCCCCCCCh
004010C6   rep stos     dword ptr [edi]
21:        int temp ;
22:
```

Context: swap(int *,

Name	Value
⊞ x	0x0012ff7c
⊞ y	0x0012ff78

```
swap(int
main(int
mainCRTSt
KERNEL32!
```

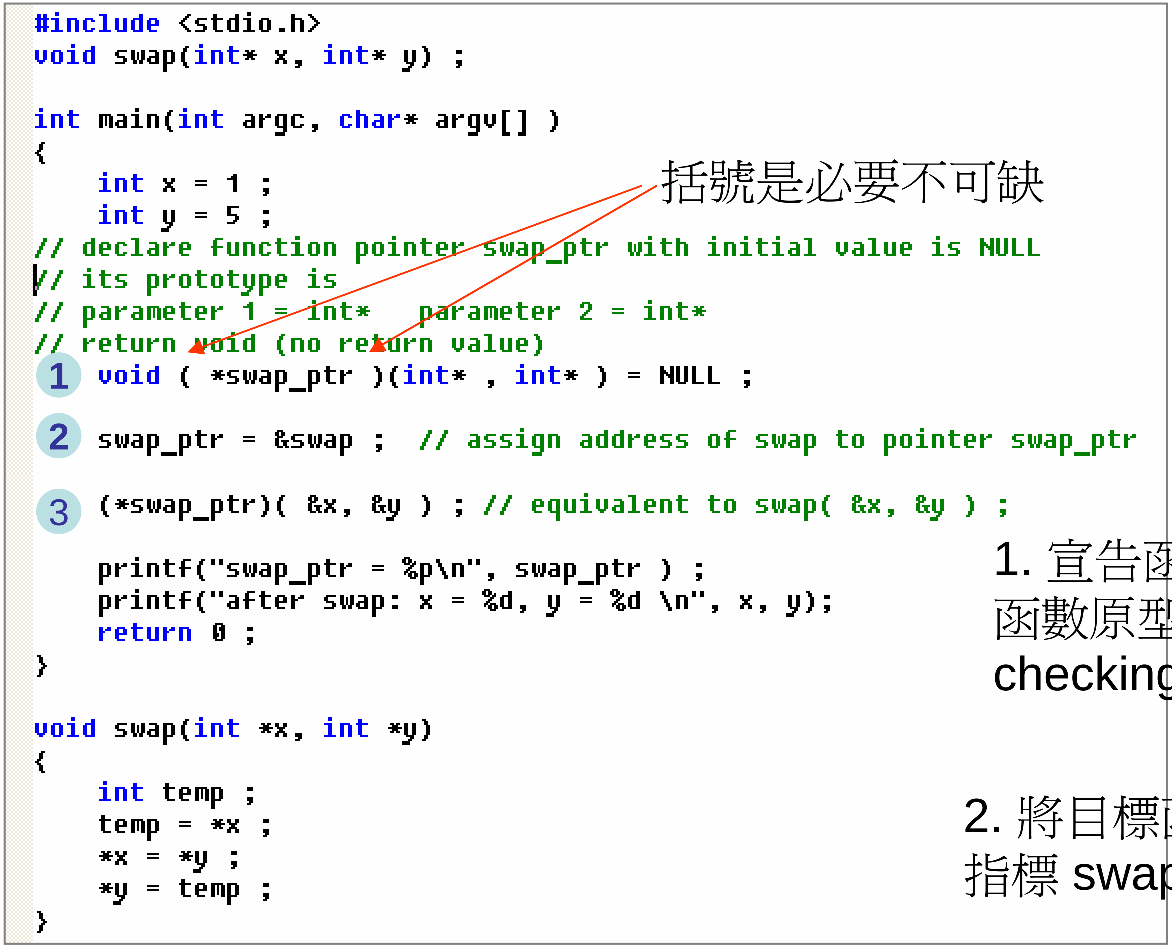
Name	Value
⊞ &main	0x00401020 "U HSVW 號■"
main	0x00401020 main(int, char * *)
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap	0x004010b0 swap(int *, int *)

進入函數 swap 後第一個指令, 其存在位址 0x004010b0 之處, 所以 swap 的位址可爲此處

Question 1: How to define function pointer?

Question 2: How to assign function pointer an address of another function?

Function pointer (函數指標) [1]



```
#include <stdio.h>
void swap(int* x, int* y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;
// declare function pointer swap_ptr with initial value is NULL
// its prototype is
// parameter 1 = int*   parameter 2 = int*
// return void (no return value)
    void ( *swap_ptr )(int* , int* ) = NULL ;

    swap_ptr = &swap ;  // assign address of swap to pointer swap_ptr

    (*swap_ptr)( &x, &y ) ; // equivalent to swap( &x, &y ) ;

    printf("swap_ptr = %p\n", swap_ptr ) ;
    printf("after swap: x = %d, y = %d \n", x, y);
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;
    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

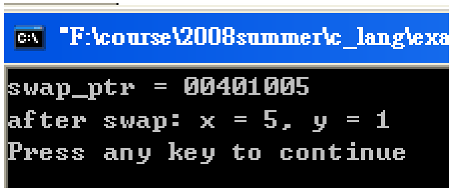
括號是必要不可缺

函數指標三部曲

1. 宣告函數指標 swap_ptr, 其函數原型宣告用來作 type checking

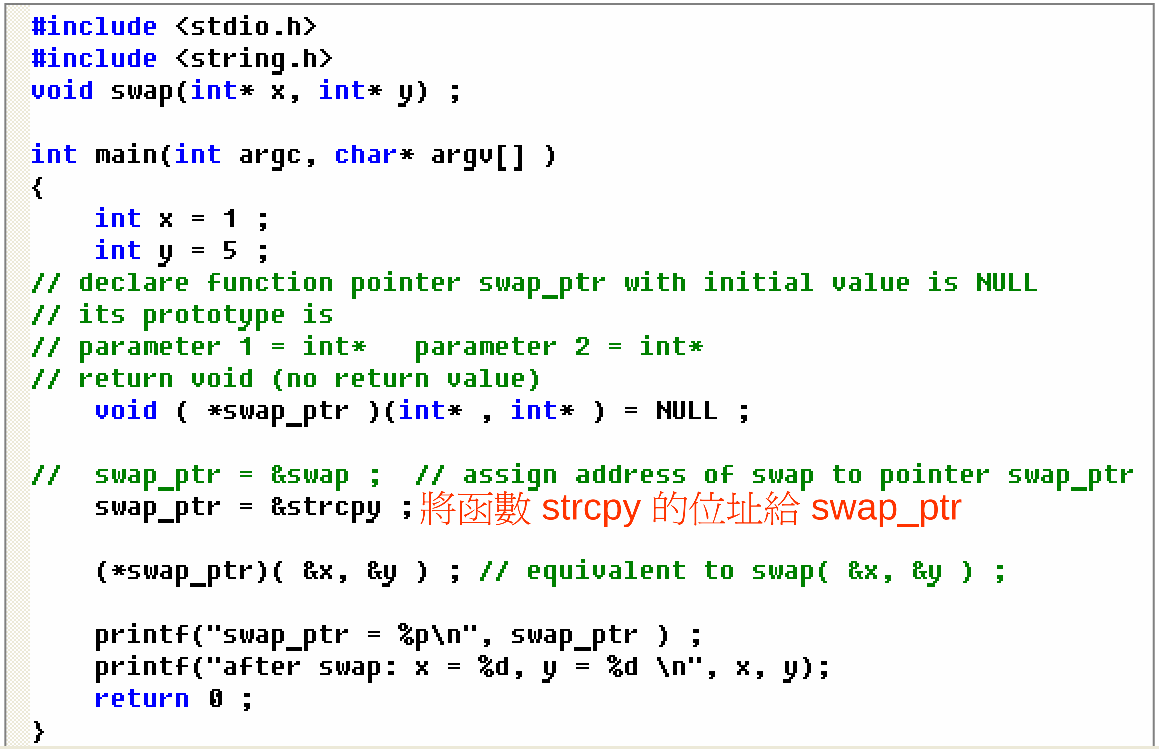
2. 將目標函數 swap 的位址存入指標 swap_ptr 內

3. 用 dereference 算子將 指標 swap_ptr 內的值視爲函數位址



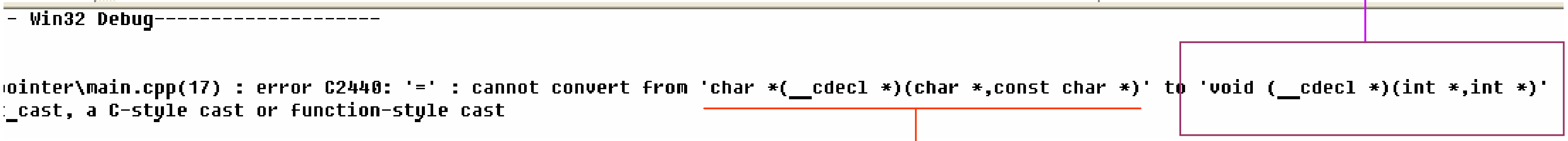
```
swap_ptr = 00401005
after swap: x = 5, y = 1
Press any key to continue
```

Function pointer (函數指標)　[2]



```
#include <stdio.h>
#include <string.h>
void swap(int* x, int* y) ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;
// declare function pointer swap_ptr with initial value is NULL
// its prototype is
// parameter 1 = int*   parameter 2 = int*
// return void (no return value)
    void ( *swap_ptr )(int* , int* ) = NULL ;

//  swap_ptr = &swap ;  // assign address of swap to pointer swap_ptr
    swap_ptr = &strcpy ;

    (*swap_ptr)( &x, &y ) ; // equivalent to swap( &x, &y ) ;

    printf("swap_ptr = %p\n", swap_ptr ) ;
    printf("after swap: x = %d, y = %d \n", x, y);
    return 0 ;
}
```

將函數 strcpy 的位址給 swap_ptr



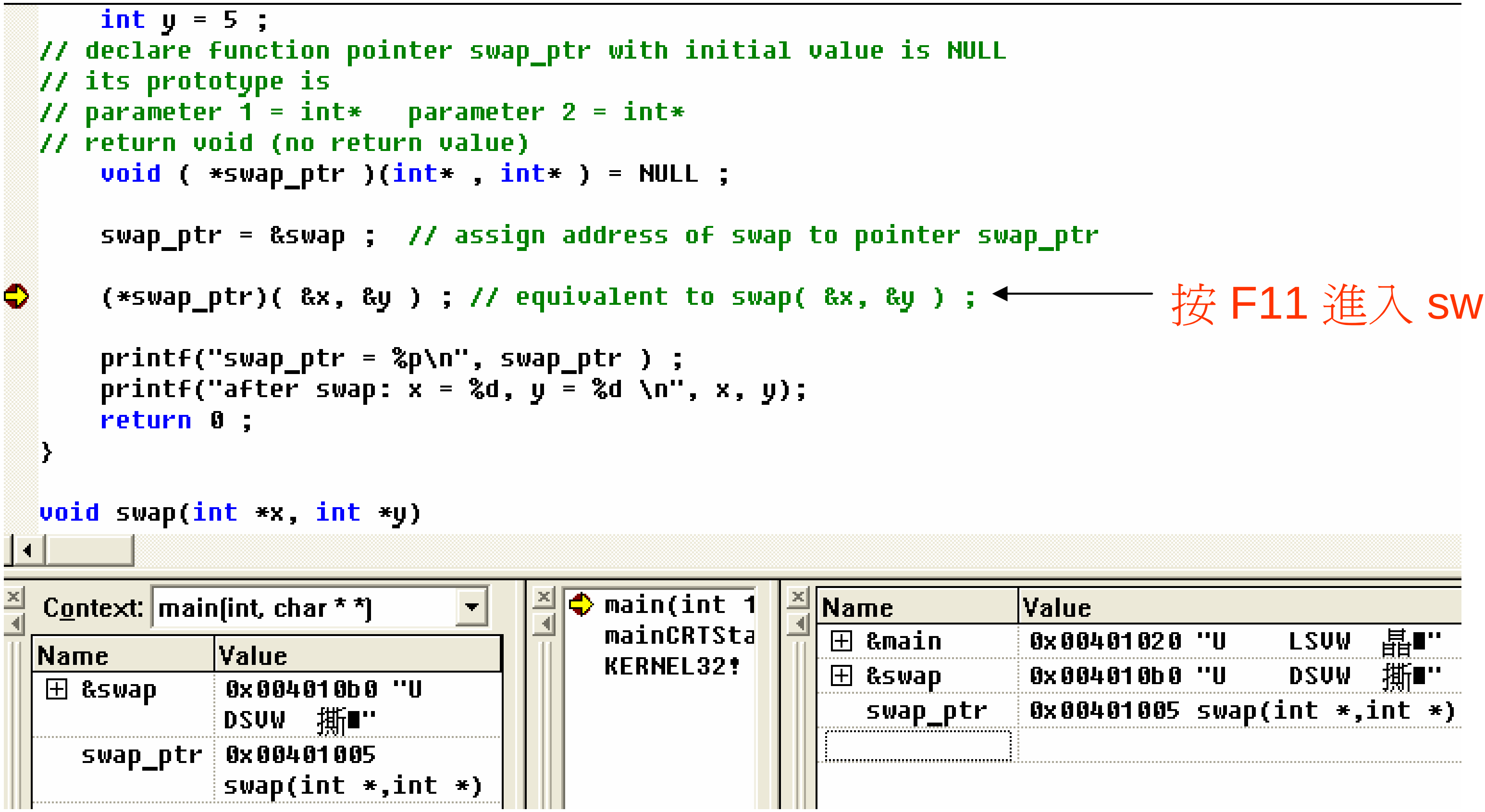
```
- Win32 Debug--------------------


ointer\main.cpp(17) : error C2440: '=' : cannot convert from 'char *(__cdecl *)(char *,const char *)' to 'void (__cdecl *)(int *,int *)'
_cast, a C-style cast or function-style cast
```

swap_ptr 的原型

strcpy 的原型爲 char* strcpy(char* , const char*)

和 swap_ptr 的原型不合

Function pointer (函數指標)　[3]



```
    int y = 5 ;
// declare function pointer swap_ptr with initial value is NULL
// its prototype is
// parameter 1 = int*   parameter 2 = int*
// return void (no return value)
    void ( *swap_ptr )(int* , int* ) = NULL ;

    swap_ptr = &swap ;  // assign address of swap to pointer swap_ptr

    (*swap_ptr)( &x, &y ) ; // equivalent to swap( &x, &y ) ;

    printf("swap_ptr = %p\n", swap_ptr ) ;
    printf("after swap: x = %d, y = %d \n", x, y);
    return 0 ;
}

void swap(int *x, int *y)
```

按 F11 進入 swap

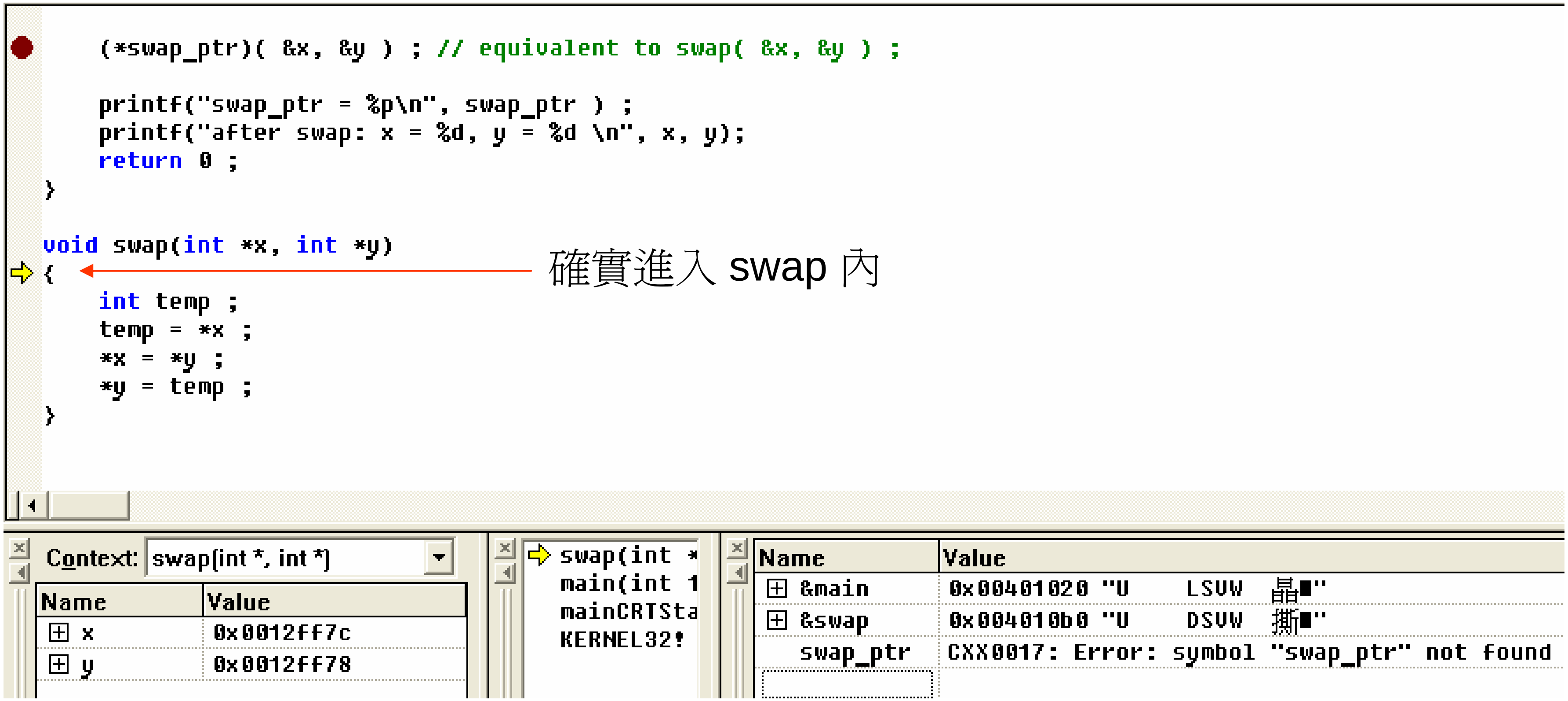
Context: main(int, char * *)

Name	Value
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap_ptr	0x00401005 swap(int *,int *)

```
main(int 1
mainCRTSta
KERNEL32!
```

Name	Value
⊞ &main	0x00401020 "U LSVW 晶■"
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap_ptr	0x00401005 swap(int *,int *)

Function pointer (函數指標)   [4]



```
    (*swap_ptr)( &x, &y ) ; // equivalent to swap( &x, &y ) ;

    printf("swap_ptr = %p\n", swap_ptr ) ;
    printf("after swap: x = %d, y = %d \n", x, y);
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;
    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

確實進入 swap 內

Context: swap(int *, int *)

Name	Value
⊞ x	0x0012ff7c
⊞ y	0x0012ff78

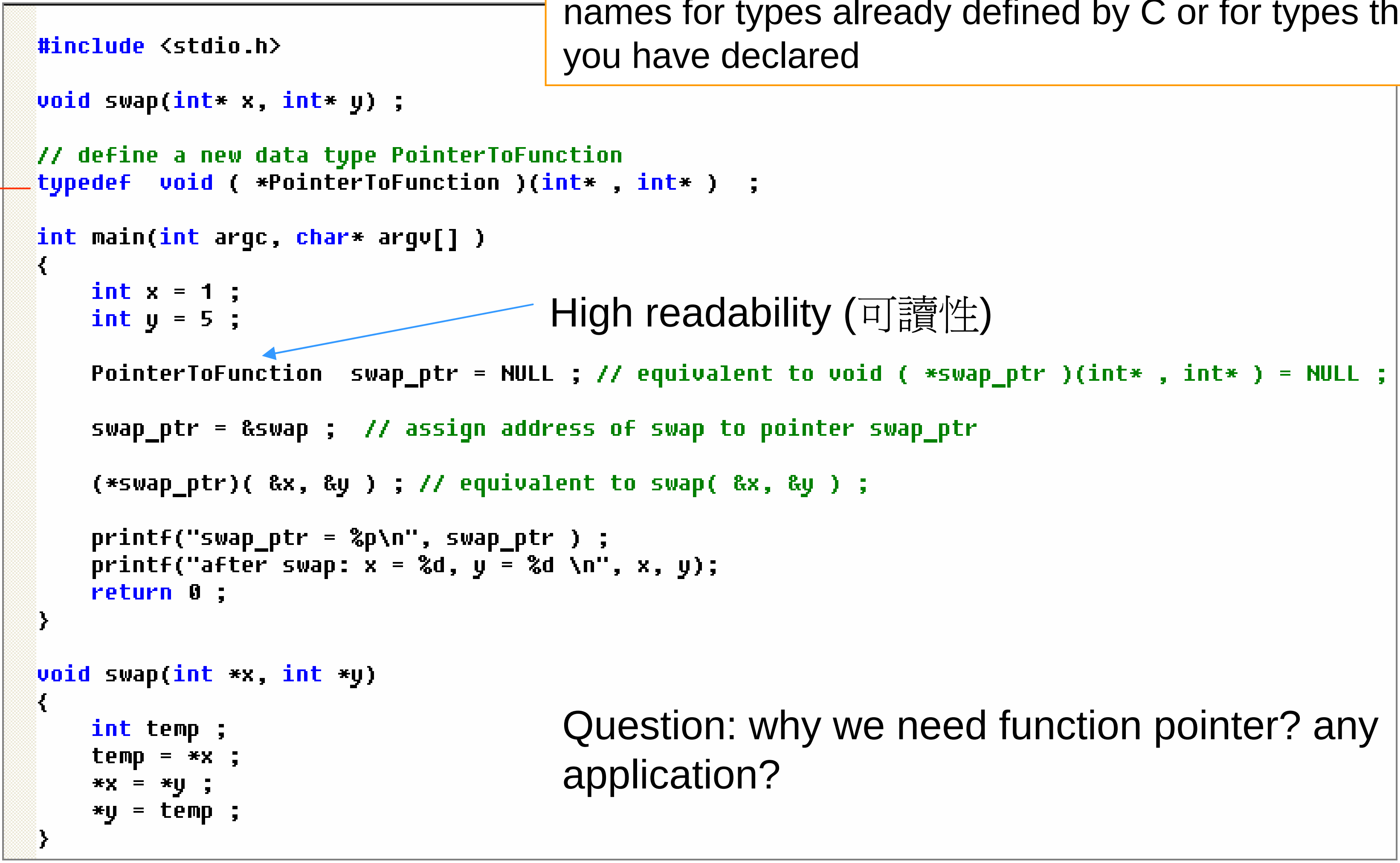
```
swap(int *
main(int 1
mainCRTSta
KERNEL32!
```

Name	Value
⊞ &main	0x00401020 "U LSVW 晶■"
⊞ &swap	0x004010b0 "U DSVW 撕■"
swap_ptr	CXX0017: Error: symbol "swap_ptr" not found

Question: declaration void (*swap_ptr)(int* , int*) is long and awkward (笨拙), can we have any other choice?

Function pointer (函數指標)   [5]

typedef defines a new type, You can use typedef declarations to construct shorter or more meaningful names for types already defined by C or for types that you have declared



```
#include <stdio.h>

void swap(int* x, int* y) ;

// define a new data type PointerToFunction
typedef  void ( *PointerToFunction )(int* , int* )  ;

int main(int argc, char* argv[] )
{
    int x = 1 ;
    int y = 5 ;

    PointerToFunction  swap_ptr = NULL ; // equivalent to void ( *swap_ptr )(int* , int* ) = NULL ;

    swap_ptr = &swap ;  // assign address of swap to pointer swap_ptr

    (*swap_ptr)( &x, &y ) ; // equivalent to swap( &x, &y ) ;

    printf("swap_ptr = %p\n", swap_ptr ) ;
    printf("after swap: x = %d, y = %d \n", x, y);
    return 0 ;
}

void swap(int *x, int *y)
{
    int temp ;
    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

High readability (可讀性)

Question: why we need function pointer? any application?

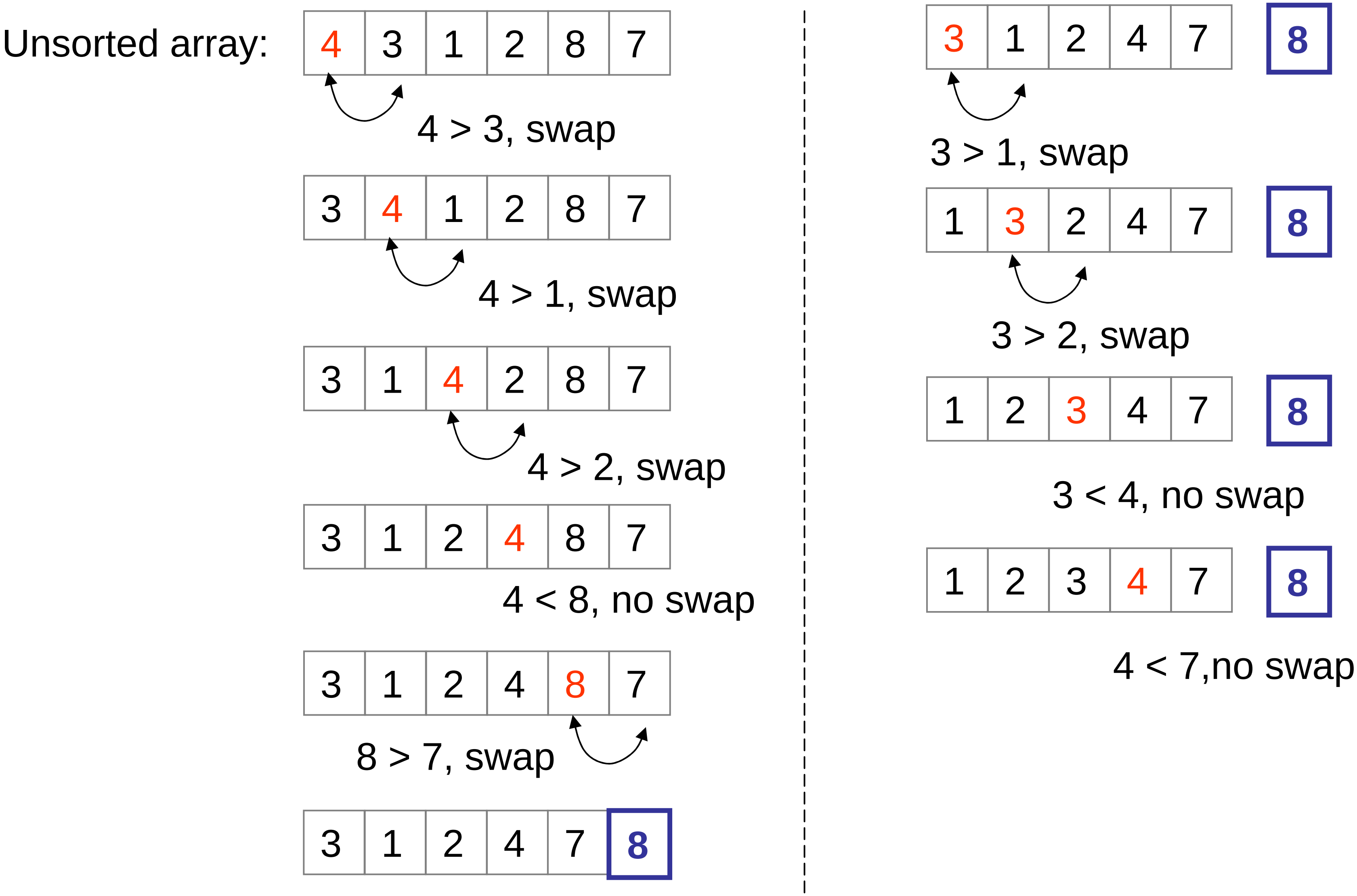
Application of function pointer: bubble sort

- bubble sort(氣泡排序法) is simplest way for sorting, the basic idea is “push the largest element upward to last location, then recursively do the same thing over remaining unsorted sub-array”.

pseudocode

```
Given un-sorted array $a[0:n]$
$for\ \ k = n : -1 : 1$
    $for\ \ j = 0 : 1 : k-1$
        if $a[j] > a[j+1]$ then $swap(a[j], a[j+1])$
    $endfor$
$endfor$
```

Process of bubble sort [1]



Unsorted array: 4 3 1 2 8 7

4 > 3, swap

3 4 1 2 8 7

4 > 1, swap

3 1 4 2 8 7

4 > 2, swap

3 1 2 4 8 7

4 < 8, no swap

3 1 2 4 8 7

8 > 7, swap

3 1 2 4 7 8

First pass, k = 5

3 1 2 4 7 8

3 > 1, swap

1 3 2 4 7 8

3 > 2, swap

1 2 3 4 7 8

3 < 4, no swap

1 2 3 4 7 8

4 < 7,no swap

second pass, k = 4

Process of bubble sort [2]

1	2	3	4	7	8

1< 2,no swap

1	2	3	4	7	8

2< 3,no swap

1	2	3	4	7	8

3< 4,no swap

third pass, k = 3

1	2	3	4	7	8

1< 2,no swap

1	2	3	4	7	8

2< 3,no swap

4-th pass, k = 2

1	2	3	4	7	8

1< 2,no swap

5-th pass, k = 1

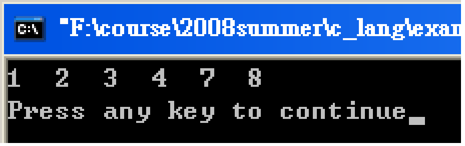
Sorted array:

1	2	3	4	7	8

bubble sort: integer version

main.cpp

```c
#include <stdio.h>

// bubble sort for integer
// definition is in bubble_sort_int.cpp
void  bubble_sort_int( int a[] , int n ) ;

int main( int argc, char* argv[] )
{
    int i ;
    int a[] = { 4, 3, 1, 2, 8, 7 } ;

    bubble_sort_int( a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%d  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



```
1  2  3  4  7  8
Press any key to continue_
```

bubble_sort_in.cpp

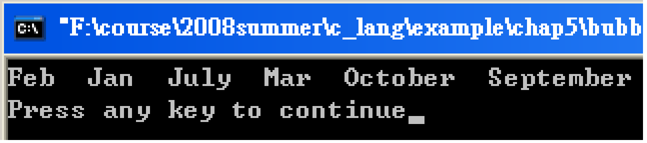
```c
void swap_int( int *x, int *y ) ;

// sort integer array a[0], a[1], ..., a[n] in ascending order
void  bubble_sort_int( int a[] , int n )
{
    int k , j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){
            if ( a[j] > a[j+1] ){
                swap_int( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_int( int *x, int *y )
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

Question: Can we sort string?

bubble sort: string version

main.cpp

```
#include <stdio.h>

// bubble sort for integer
// definition is in bubble_sort_int.cpp
void  bubble_sort_int( int a[] , int n ) ;

// bubble sort for string
// definition is in bubble_sort_string.cpp
void  bubble_sort_string( char* a[] , int n ) ;

int main( int argc, char* argv[] )
{
    int i ;
    char* a[] = { "September", "Jan" , "Mar",
                  "Feb"      , "July", "October" } ;

    bubble_sort_string( a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%s  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



```
Feb  Jan  July  Mar  October  September
Press any key to continue_
```

bubble_sort_string.cpp

```
#include <string.h>

void swap_string( char **x, char **y ) ;

// sort string array a[0], a[1], ..., a[n] in
// ascending order, compare two strings with
// lexicographic order
void  bubble_sort_string( char* a[] , int n )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if ( strcmp( a[j], a[j+1] ) > 0  ){

                swap_string( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_string( char **x, char **y )
{
    char* temp ;
    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

x is a pointer pointing to data type char* (string)

swap_string only swaps pointers, not string content

Process of bubble sort : string [1]

Unsorted array:

September	Jan	Mar	Feb	July	October
0x00422fdc	0x00422048	0x00422044	0x00422040	0x00422fd4	0x00422034

September > Jan, swap

Jan	September	Mar	Feb	July	October
0x00422048	0x00422fdc	0x00422044	0x00422040	0x00422fd4	0x00422034

September > Mar, swap

Jan	Mar	September	Feb	July	October
0x00422048	0x00422044	0x00422fdc	0x00422040	0x00422fd4	0x00422034

September > Feb, swap

Jan	Mar	Feb	September	July	October
0x00422048	0x00422044	0x00422040	0x00422fdc	0x00422fd4	0x00422034

September > July, swap

Process of bubble sort : string [2]

Jan	Mar	Feb	July	September	October
0x00422048	0x00422044	0x00422040	0x00422fd4	0x00422fdc	0x00422034

September > October, swap

Jan	Mar	Feb	July	October	September
0x00422048	0x00422044	0x00422040	0x00422fd4	0x00422034	0x00422fdc

Jan < Mar, no swap

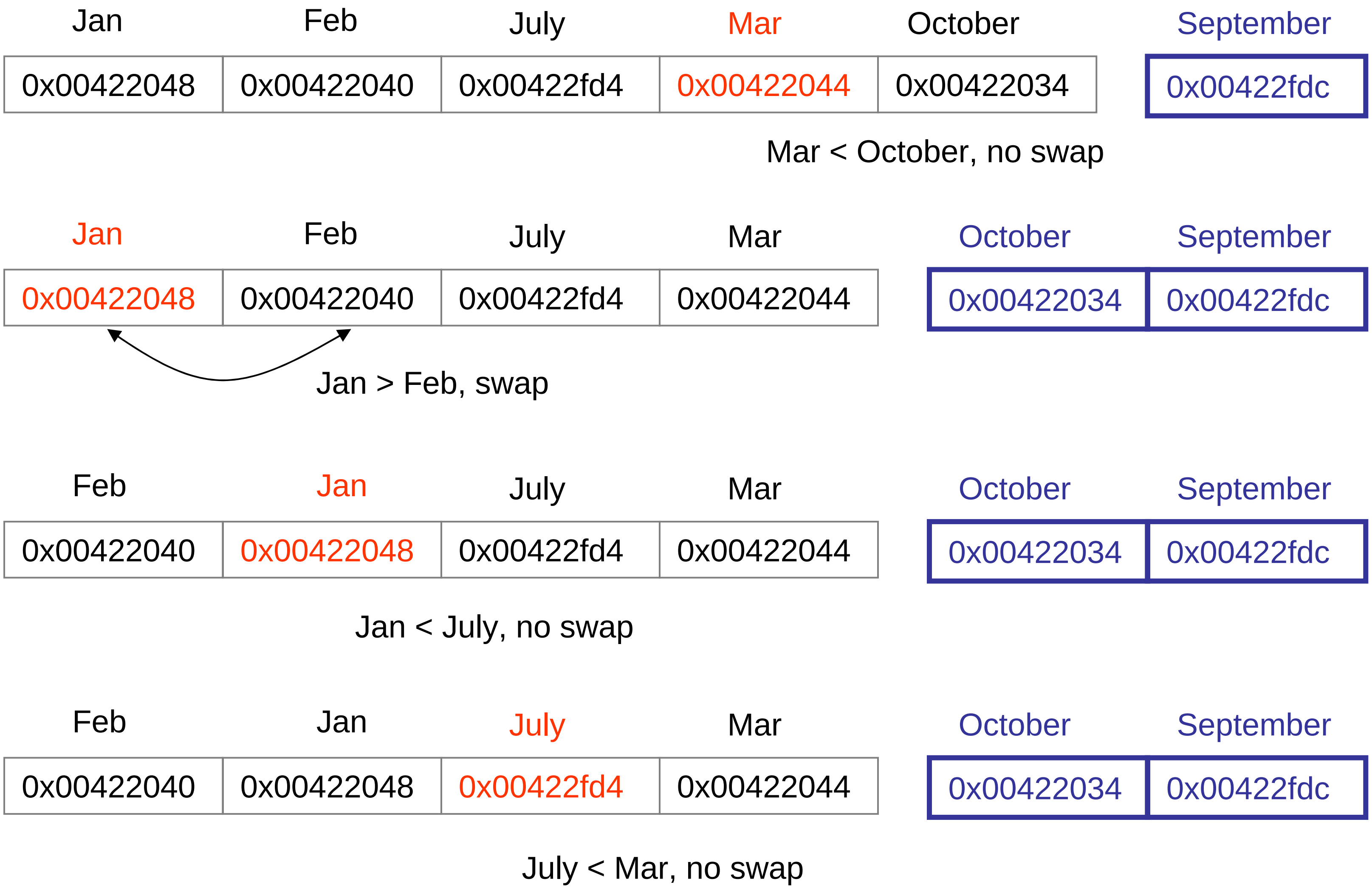
Jan	Mar	Feb	July	October	September
0x00422048	0x00422044	0x00422040	0x00422fd4	0x00422034	0x00422fdc

Mar > Feb, swap

Jan	Feb	Mar	July	October	September
0x00422048	0x00422040	0x00422044	0x00422fd4	0x00422034	0x00422fdc

Mar > July, swap

Process of bubble sort : string [3]



Jan	Feb	July	Mar	October	September
0x00422048	0x00422040	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Mar < October, no swap

Jan	Feb	July	Mar	October	September
0x00422048	0x00422040	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Jan > Feb, swap

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Jan < July, no swap

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

July < Mar, no swap

Process of bubble sort : string [4]

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Feb < Jan, no swap

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Jan < July, no swap

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

Feb < Jan, no swap

Feb	Jan	July	Mar	October	September
0x00422040	0x00422048	0x00422fd4	0x00422044	0x00422034	0x00422fdc

sorting complete

observation

- Data type is immaterial, we only need to provide comparison operator. In other words, framework of bubble sort is independent of comparison operation.
- How can we implement an algorithm for bubble sort such that it is independent of data type, for example, string ?

pseudocode

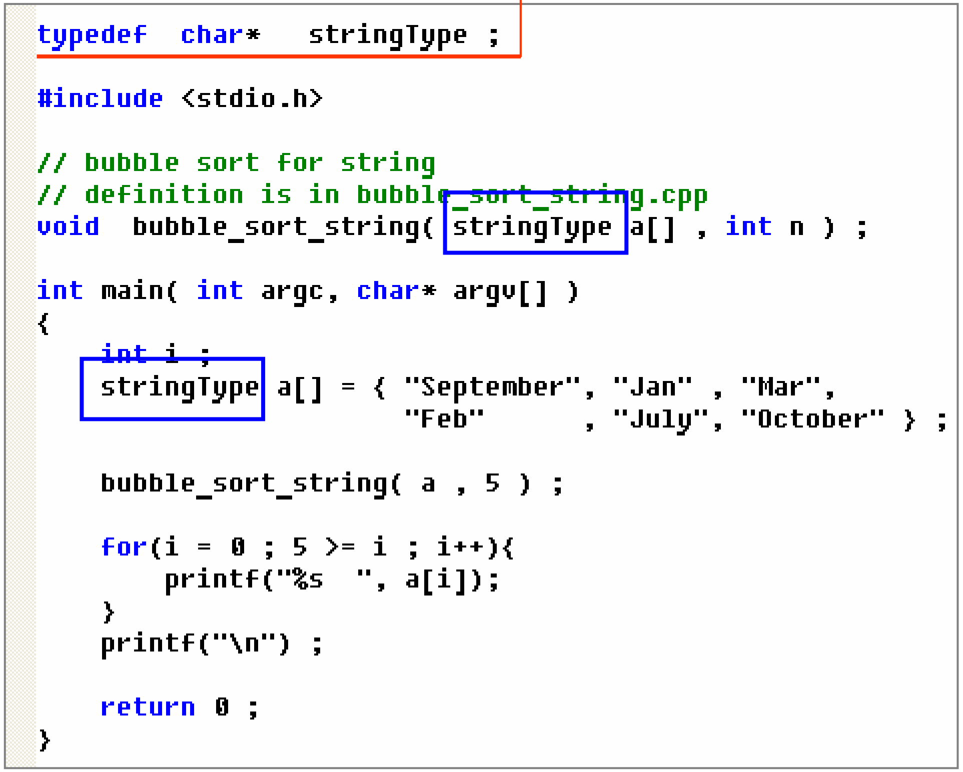
```
Given un-sorted array $a[0:n]$
$for \;\; k = n : -1 : 1$
    $for \;\; j = 0 : 1 : k-1$
        if $a[j] > a[j+1]$ then $swap(a[j], a[j+1])$
    $endfor$
$endfor$
```

User-defined comparison operator

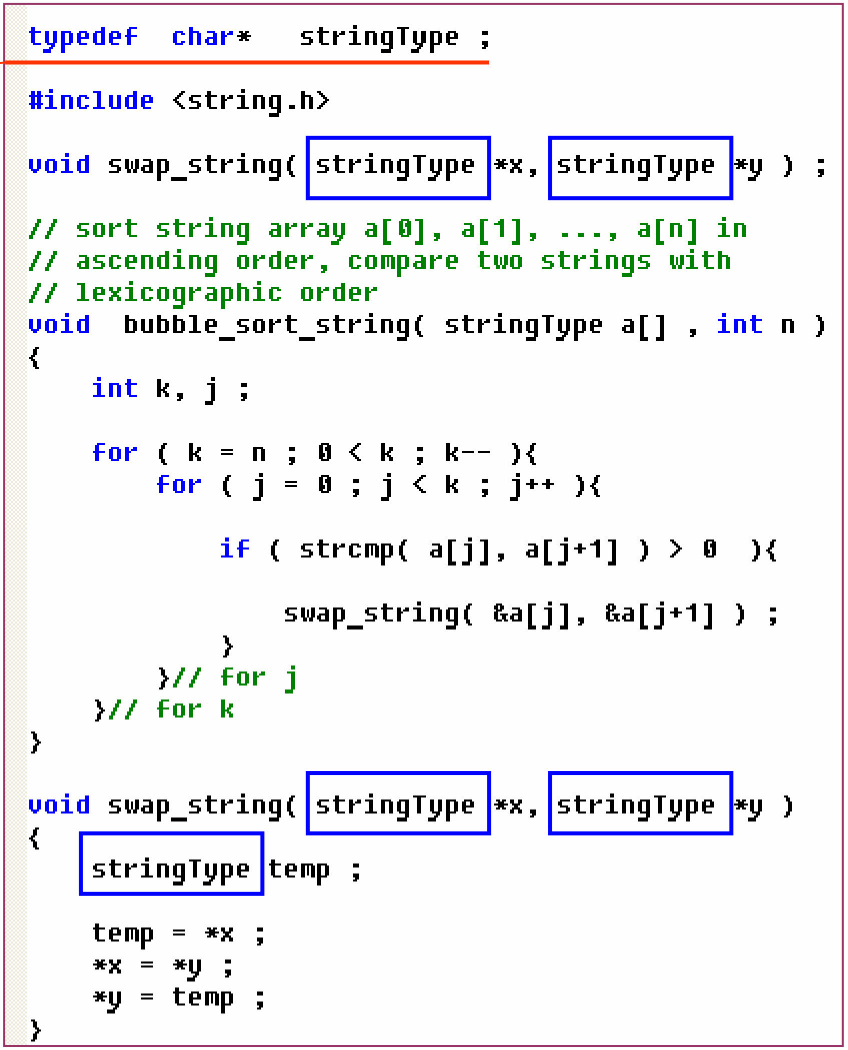
Framework of bubble sort [1]

define a new data type

string_type = char*



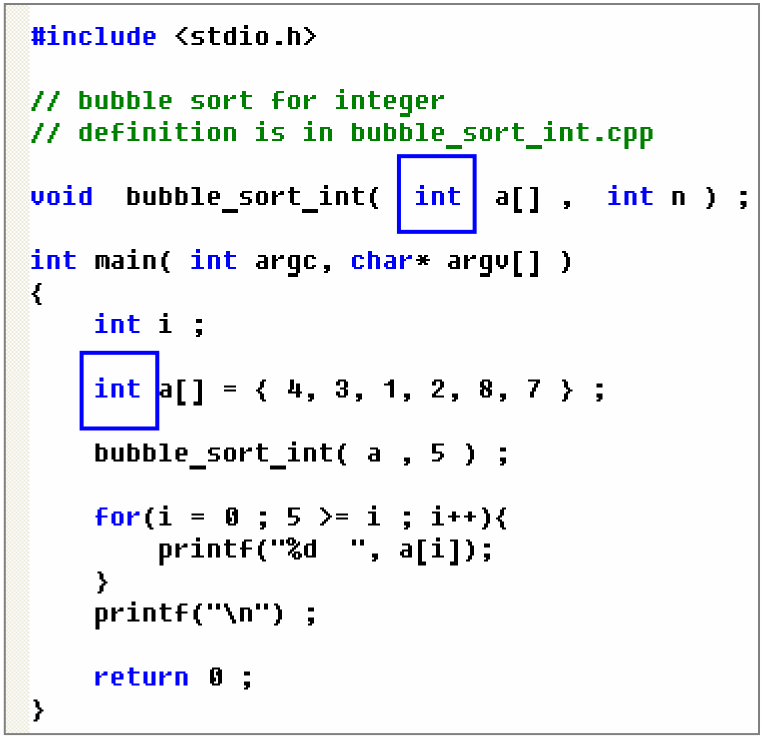
```
typedef  char*   stringType ;

#include <stdio.h>

// bubble sort for string
// definition is in bubble_sort_string.cpp
void  bubble_sort_string( stringType a[] , int n ) ;

int main( int argc, char* argv[] )
{
    int i ;
    stringType a[] = { "September", "Jan"  , "Mar",
                       "Feb"       , "July", "October" } ;

    bubble_sort_string( a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%s  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



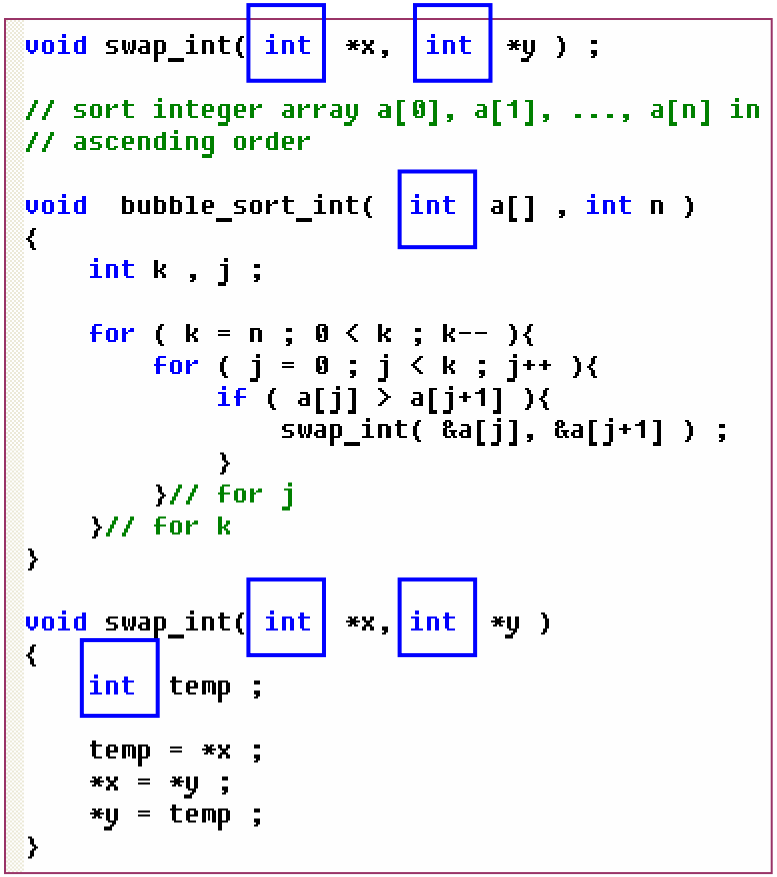
```
typedef  char*   stringType ;

#include <string.h>

void swap_string( stringType *x, stringType *y ) ;

// sort string array a[0], a[1], ..., a[n] in
// ascending order, compare two strings with
// lexicographic order
void  bubble_sort_string( stringType a[] , int n )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if ( strcmp( a[j], a[j+1] ) > 0  ){

                swap_string( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_string( stringType *x, stringType *y )
{
    stringType temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

□ Data type for sorting

Framework of bubble sort   [2]



```c
#include <stdio.h>

// bubble sort for integer
// definition is in bubble_sort_int.cpp

void  bubble_sort_int( int a[] ,  int n ) ;

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

    int a[] = { 4, 3, 1, 2, 8, 7 } ;

    bubble_sort_int( a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%d  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



```c
void swap_int( int *x, int *y ) ;

// sort integer array a[0], a[1], ..., a[n] in
// ascending order

void  bubble_sort_int( int a[] , int n )
{
    int k , j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){
            if ( a[j] > a[j+1] ){
                swap_int( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_int( int *x, int *y )
{
    int temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

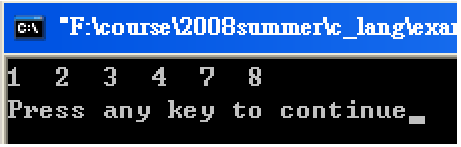
□ Data type for sorting

Question: How about if we change int to void* ?

Framework of bubble sort   [3]

```
#include <stdio.h>

// bubble sort for integer
// definition is in bubble_sort_int.cpp

void  bubble_sort_int( void* a[] , int n ) ;

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

    int a[] = { 4, 3, 1, 2, 8, 7 } ;

    bubble_sort_int( (void**) a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%d  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

強制轉型, type checking



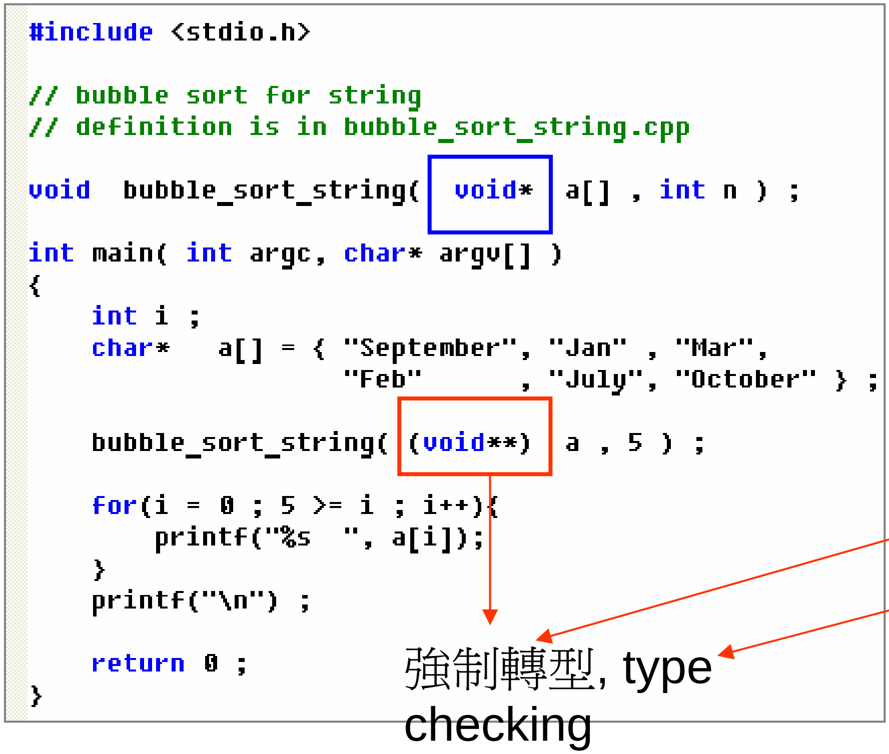
"F:\course\2008summer\c_lang\exa

```
1  2  3  4  7  8
Press any key to continue_
```

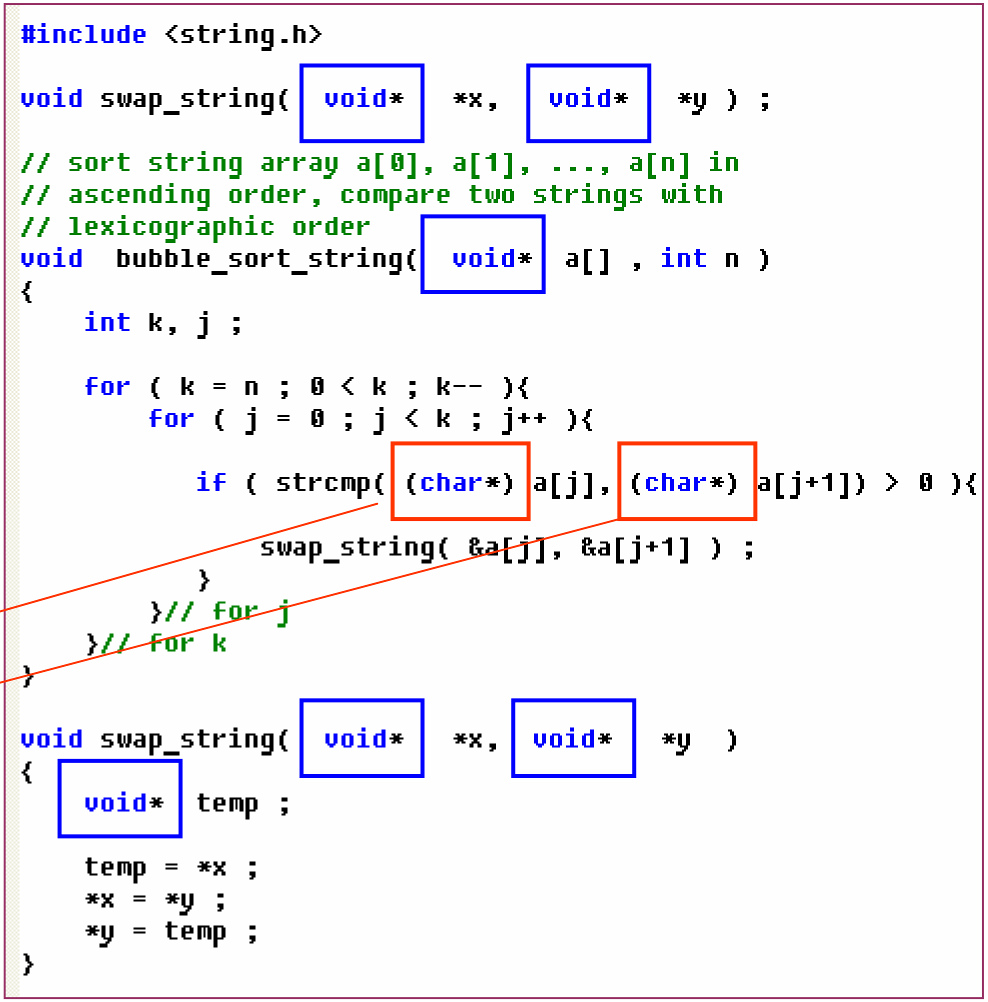
```
void swap_int( void* *x, void* *y ) ;

// sort integer array a[0], a[1], ..., a[n] in
// ascending order

void  bubble_sort_int( void* a[] , int n )
{
    int k , j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){
            if ( a[j] > a[j+1] ){
                swap_int( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_int( void* *x, void* *y )
{
    void* temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

int 和 void* 有同樣的大小,
在32位元機器上, 其size = 4 Bytes

Framework of bubble sort   [4]



```
#include <stdio.h>

// bubble sort for string
// definition is in bubble_sort_string.cpp

void  bubble_sort_string( void* a[] , int n ) ;

int main( int argc, char* argv[] )
{
    int i ;
    char*   a[] = { "September", "Jan"  , "Mar",
                    "Feb"       , "July", "October" } ;

    bubble_sort_string( (void**) a , 5 ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%s  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



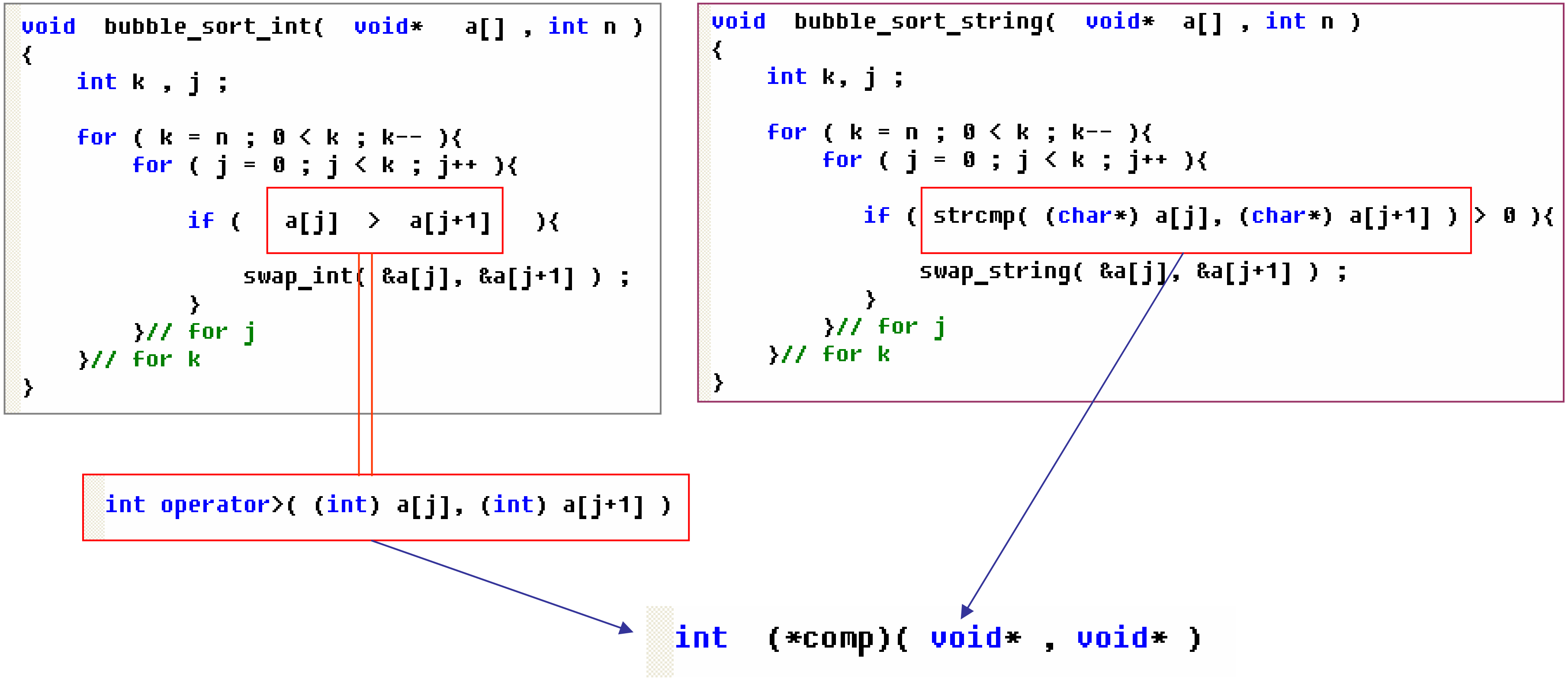
```
#include <string.h>

void swap_string( void* *x, void* *y ) ;

// sort string array a[0], a[1], ..., a[n] in
// ascending order, compare two strings with
// lexicographic order
void  bubble_sort_string( void* a[] , int n )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){
            if ( strcmp( (char*) a[j], (char*) a[j+1]) > 0 ){
                swap_string( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap_string( void* *x, void* *y  )
{
    void* temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

強制轉型, type checking

Great! swap_string is the same as swap_int.

Question: How to handle comparison operation (use function pointer) ?

Framework of bubble sort   [5]



```
void  bubble_sort_int(  void*   a[] , int n )
{
    int k , j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if (  a[j]  >  a[j+1]   ){

                swap_int( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}
```

```
void  bubble_sort_string(  void*  a[] , int n )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if ( strcmp( (char*) a[j], (char*) a[j+1] ) > 0 ){

                swap_string( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}
```

```
int operator>( (int) a[j], (int) a[j+1] )
```

```
int  (*comp)( void* , void* )
```

comp is a function pointer, either strcmp or integer operator>

$$protocol\,(\text{協定}):\quad (*comp)(s,t)\ \text{return}\ \begin{cases} <0 & \text{if}\ \ s<t \\ =0 & \text{if}\ \ s=t \\ >0 & \text{if}\ \ s>t \end{cases}$$

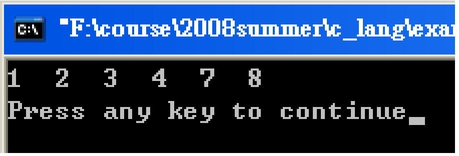
Framework of bubble sort   [6]

```
#include <stdio.h>

void  bubble_sort_prototype(  void*  a[] , int n,

                int (*comp)(void*, void*)   ) ;
// return  > 0  if x > y
//         = 0  if x = y
//         < 0  if x < y
int  integer_comp( int x, int y )
{
    return x - y ;
}

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

    int a[] = { 4, 3, 1, 2, 8, 7 } ;

    bubble_sort_prototype( (void**) a, 5,

        (int (*)(void*, void*))  &integer_comp   ) ;


    for(i = 0 ; 5 >= i ; i++){
        printf("%d  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

prototype of this function pointer

```
void swap(  void*   *x,  void*   *y  ) ;

// prototype of bubble sort, accept function pointer
// comp as comparator
void  bubble_sort_prototype(  void*  a[] , int n,

                int (*comp)(void*, void*)   )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if ( (*comp)( a[j], a[j+1] )  > 0 ){

                swap( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap(  void*   *x,  void*   *y  )
{
    void*  temp ;

    temp = *x ;
    *x = *y ;
    *y = temp ;
}
```

comp: function pointer



```
1  2  3  4  7  8
Press any key to continue_
```

Framework of bubble sort [7]

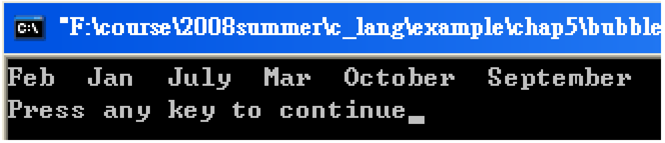
```
#include <stdio.h>
#include <string.h>

void  bubble_sort_prototype(  void*  a[] , int n,
                int (*comp)(void*, void*)   ) ;

int main( int argc, char* argv[] )
{
    int i ;
    char*   a[] = { "September", "Jan" , "Mar",
                    "Feb"       , "July", "October" } ;

    bubble_sort_prototype( (void**)  a, 5,

        (int (*)(void*, void*))  &strcmp   ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%s  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

Standard library

prototype of this function pointer

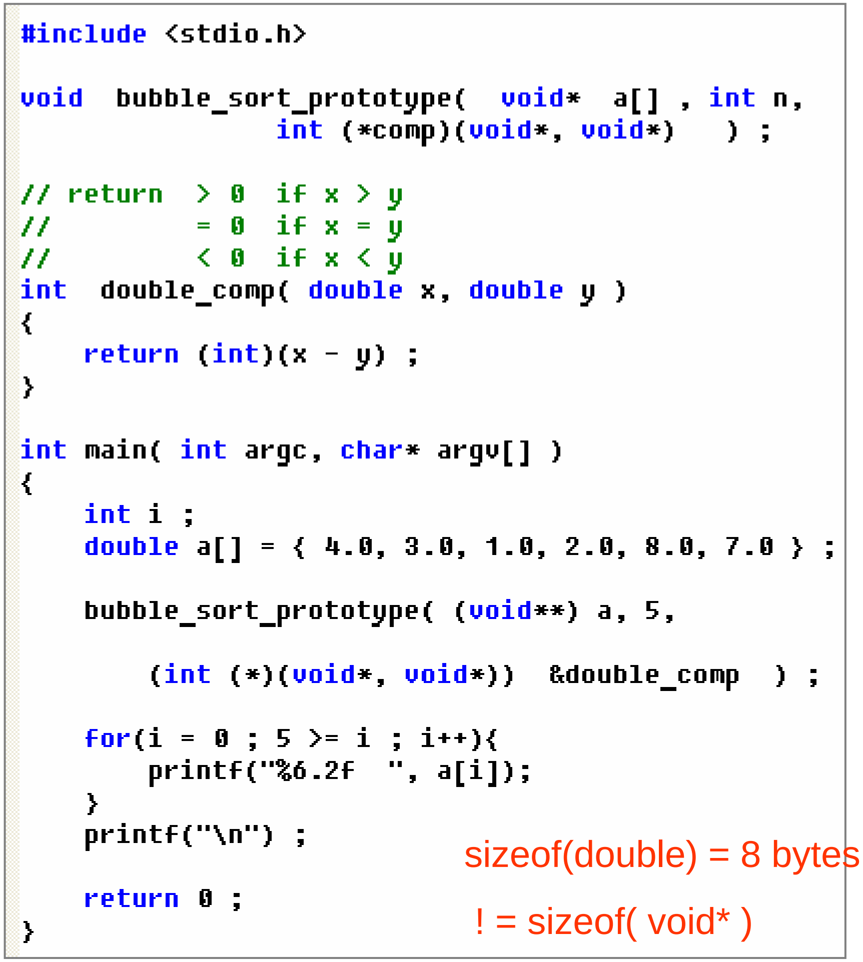
```
void swap(  void*   **x,  void*   **y  ) ;

// prototype of bubble sort, accept function pointer
// comp as comparator
void  bubble_sort_prototype(  void*  a[] , int n,

                int (*comp)(void*, void*)   )
{
    int k, j ;

    for ( k = n ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){

            if ( (*comp)( a[j], a[j+1] )  > 0 ){

                swap( &a[j], &a[j+1] ) ;
            }
        }// for j
    }// for k
}

void swap(  void*   **x,  void*   **y  )
{
    void*  temp ;

    temp = **x ;
    **x = **y ;
    **y = temp ;
}
```

Question: how about floating-point array?



```
Feb   Jan   July   Mar   October   September
Press any key to continue_
```

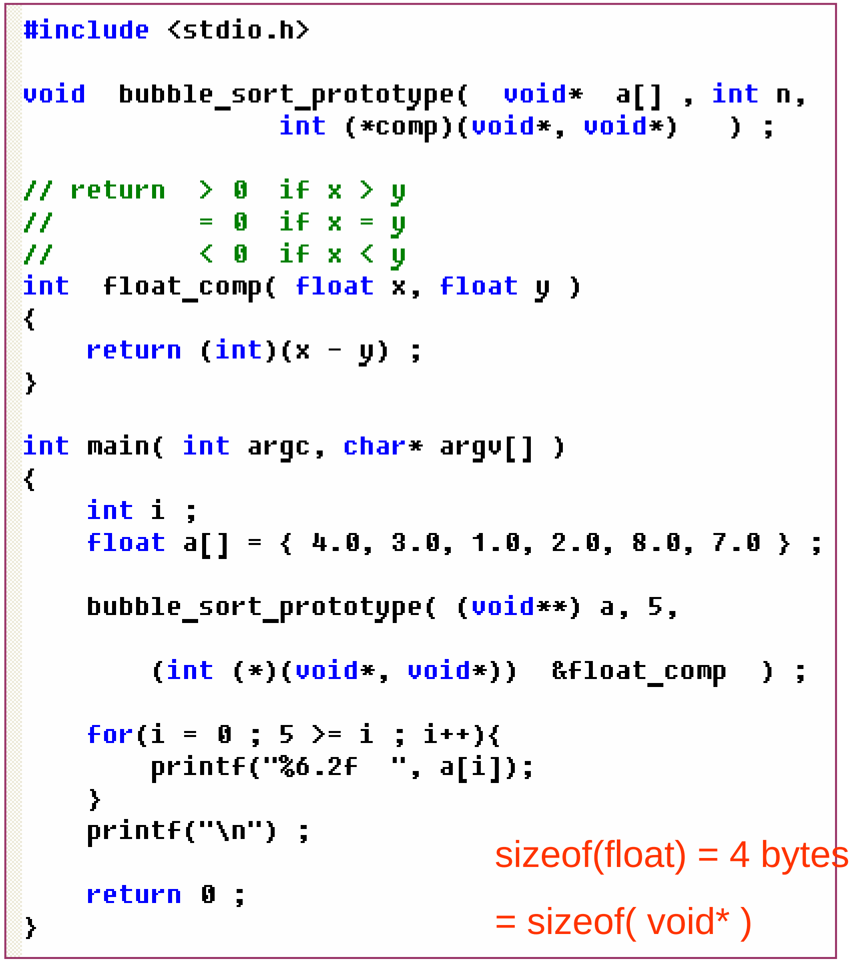
Framework of bubble sort [8]



```c
#include <stdio.h>

void  bubble_sort_prototype(  void*  a[] , int n,
                int (*comp)(void*, void*)   ) ;

// return  > 0  if x > y
//         = 0  if x = y
//         < 0  if x < y
int  double_comp( double x, double y )
{
    return (int)(x - y) ;
}

int main( int argc, char* argv[] )
{
    int i ;
    double a[] = { 4.0, 3.0, 1.0, 2.0, 8.0, 7.0 } ;

    bubble_sort_prototype( (void**) a, 5,

        (int (*)(void*, void*))  &double_comp  ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%6.2f  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

sizeof(double) = 8 bytes

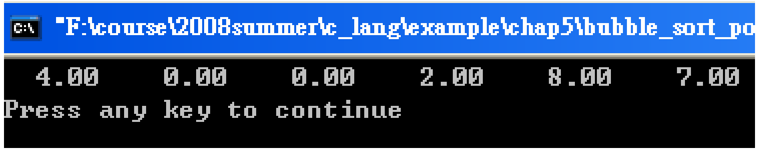
! = sizeof(void*)



```c
#include <stdio.h>

void  bubble_sort_prototype(  void*  a[] , int n,
                int (*comp)(void*, void*)   ) ;

// return  > 0  if x > y
//         = 0  if x = y
//         < 0  if x < y
int  float_comp( float x, float y )
{
    return (int)(x - y) ;
}

int main( int argc, char* argv[] )
{
    int i ;
    float a[] = { 4.0, 3.0, 1.0, 2.0, 8.0, 7.0 } ;

    bubble_sort_prototype( (void**) a, 5,

        (int (*)(void*, void*))  &float_comp  ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%6.2f  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

sizeof(float) = 4 bytes

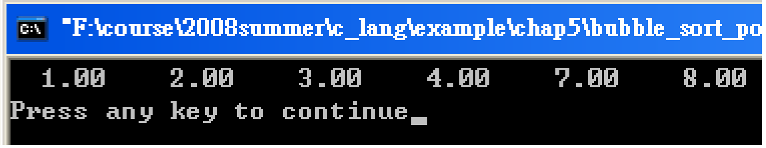
= sizeof(void*)



"F:\course\2008summer\c_lang\example\chap5\bubble_sort_po

```
  4.00    0.00    0.00    2.00    8.00    7.00
Press any key to continue
```

Wrong!



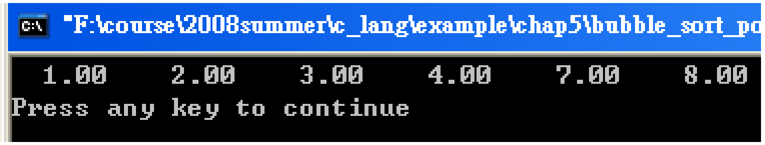
"F:\course\2008summer\c_lang\example\chap5\bubble_sort_po

```
  1.00    2.00    3.00    4.00    7.00    8.00
Press any key to continue_
```

Correct!

Framework of bubble sort [9]

```c
#include <stdio.h>

void  bubble_sort( void*  base, size_t n, size_t size,
                   int (*comp)(void*, void*)   ) ;

int  double_comp( double *x, double *y )
{
    return (int)(*x - *y) ;
}

int main( int argc, char* argv[] )
{
    int i ;
    double a[] = { 4.0, 3.0, 1.0, 2.0, 8.0, 7.0 } ;

    bubble_sort( (void*) a, 6, sizeof(double),

        (int (*)(void*, void*))  &double_comp  ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%6.2f  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```



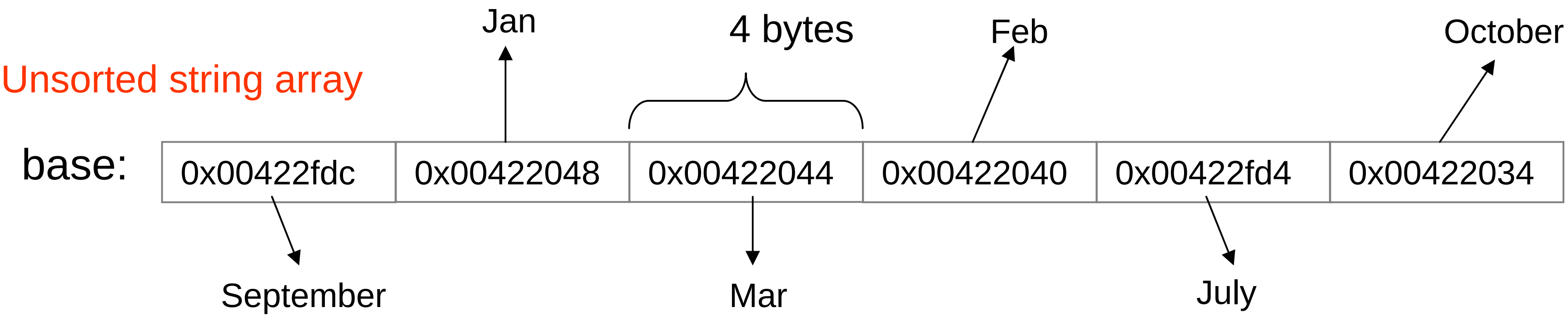
```
  1.00    2.00    3.00    4.00    7.00    8.00
Press any key to continue
```

此處 void 可視爲 any data type

```c
#include <stddef.h>
void swap( void  *x, void  *y , size_t size ) ;

// sort base[0], ... base[n-1], total n elements
void  bubble_sort( void* base, size_t n, size_t size,
                   int (*comp)(void*, void*)   )
{
    int k, j ;
    char*  a = (char*) base ;
    char*  aj ;       // &a[j]
    char*  aj_plus1 ; // &a[j+1]

    for ( k = n-1 ; 0 < k ; k-- ){
        for ( j = 0 ; j < k ; j++ ){
            aj       = a + size*j  ;
            aj_plus1 = aj + size ;
            if ( (*comp)( aj,  aj_plus1 )  > 0 ){
                swap( (void*) aj, (void*) aj_plus1,
                      size ) ;
            }
        }// for j
    }// for k
}
void swap(  void *x,  void *y, size_t size  )
{
    size_t  i ;
    char temp ;
    char*  px = (char*) x ;
    char*  py = (char*) y ;

    for ( i = 0 ; i < size ; i++){
        temp = *px ;
        *px = *py ;
        *py = temp ;
        px++ ;
        py++ ;
    }
}
```

Framework of bubble sort [10]

Unsorted string array



Jan

4 bytes

Feb

October

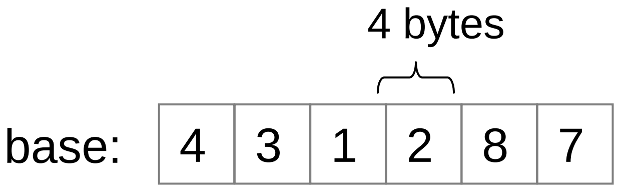
base:	0x00422fdc	0x00422048	0x00422044	0x00422040	0x00422fd4	0x00422034

September

Mar

July

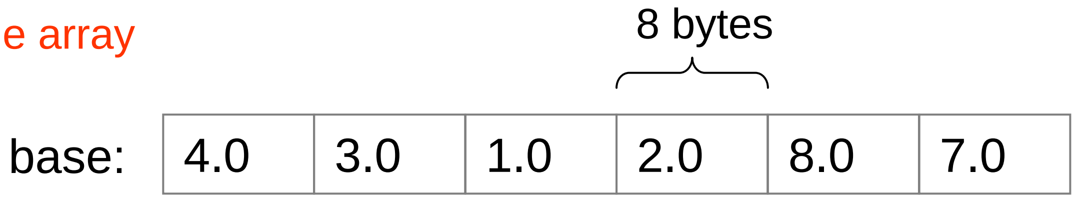
Unsorted integer array



4 bytes

base:	4	3	1	2	8	7

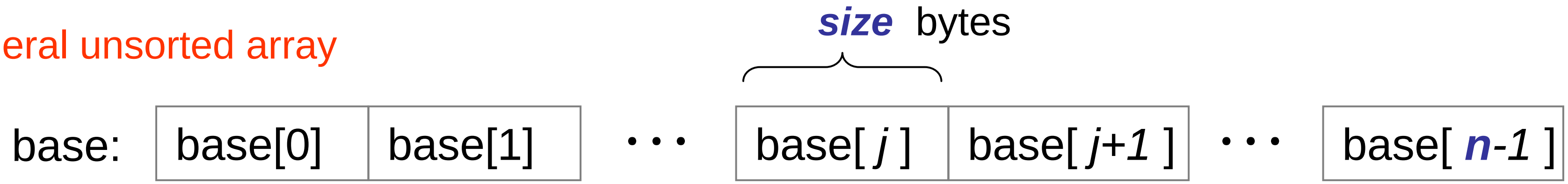
Unsorted double array



8 bytes

base:	4.0	3.0	1.0	2.0	8.0	7.0

General unsorted array

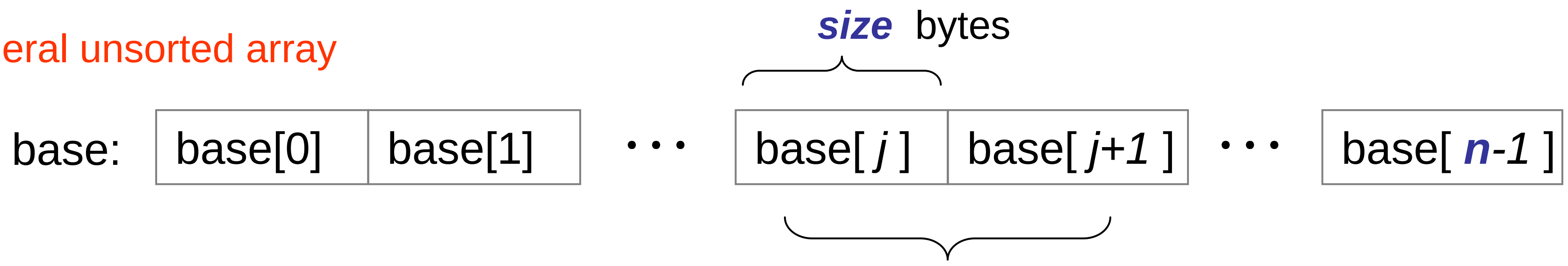


size bytes

base:	base[0]	base[1]	· · ·	base[*j*]	base[*j+1*]	· · ·	base[***n***-*1*]

Framework of bubble sort [11]

General unsorted array

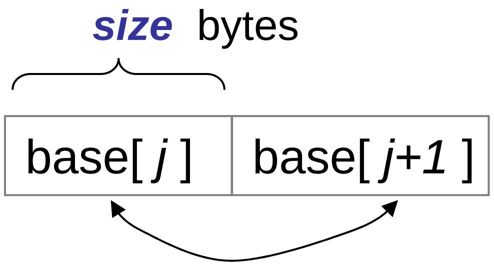


size bytes

base: | base[0] | base[1] | · · · | base[*j*] | base[*j+1*] | · · · | base[***n***-*1*]

(***cmp**) (&base[*j*], &base[*j-1*])

We send address of base[*j*] and base[*j-1*] to user-defined comparison operator ***cmp** since we don’t know the data type so far.



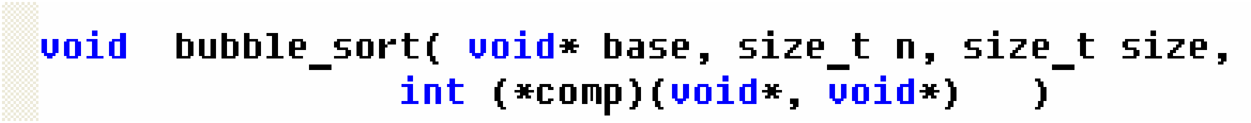
size bytes

base[*j*] | base[*j+1*]

swap(&base[*j*], &base[*j-1*], ***size***)

We send address of base[*j*] and base[*j-1*] to swap function since we don’t know the data type, also in order to swap both elements we require ***size*** of data type

Framework of bubble sort [12]



```
void  bubble_sort( void* base, size_t n, size_t size,
                   int (*comp)(void*, void*)   )
```

void **qsort**(void ***_base_**, size_t ***n***, size_t ***size***,

int (****cmp***)(const void *, const void *))

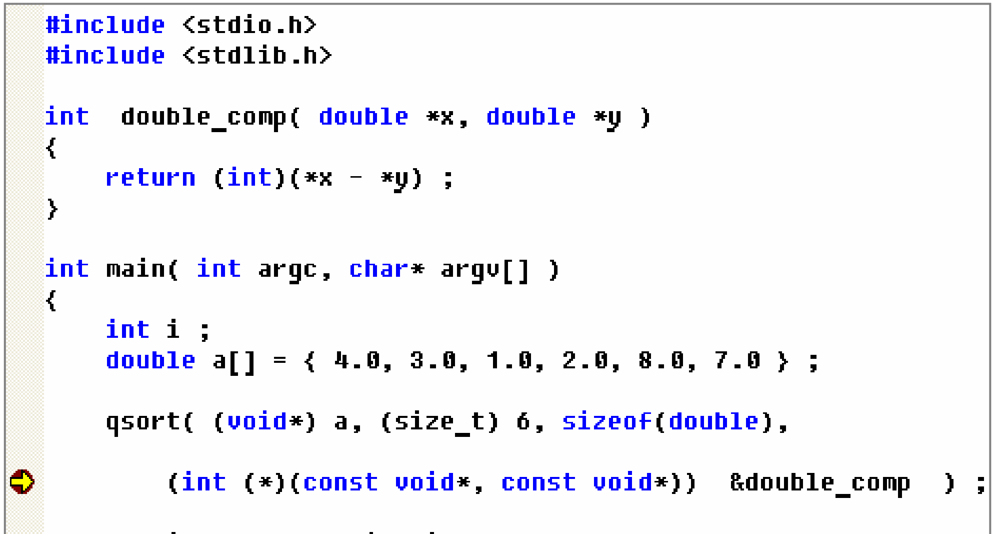
page 253

qsort sorts into ascending order an array ***base***[0], … ***base***[***n*** -1] of objects of size ***size***. The comparison function ***cmp*** is

$$protocol\left(\text{協定}\right):\quad \left(*cmp\right)\left(s,t\right)\text{ return }\begin{cases} <0 \text{ if } s<t \\ =0 \text{ if } s=t \\ >0 \text{ if } s>t \end{cases}$$

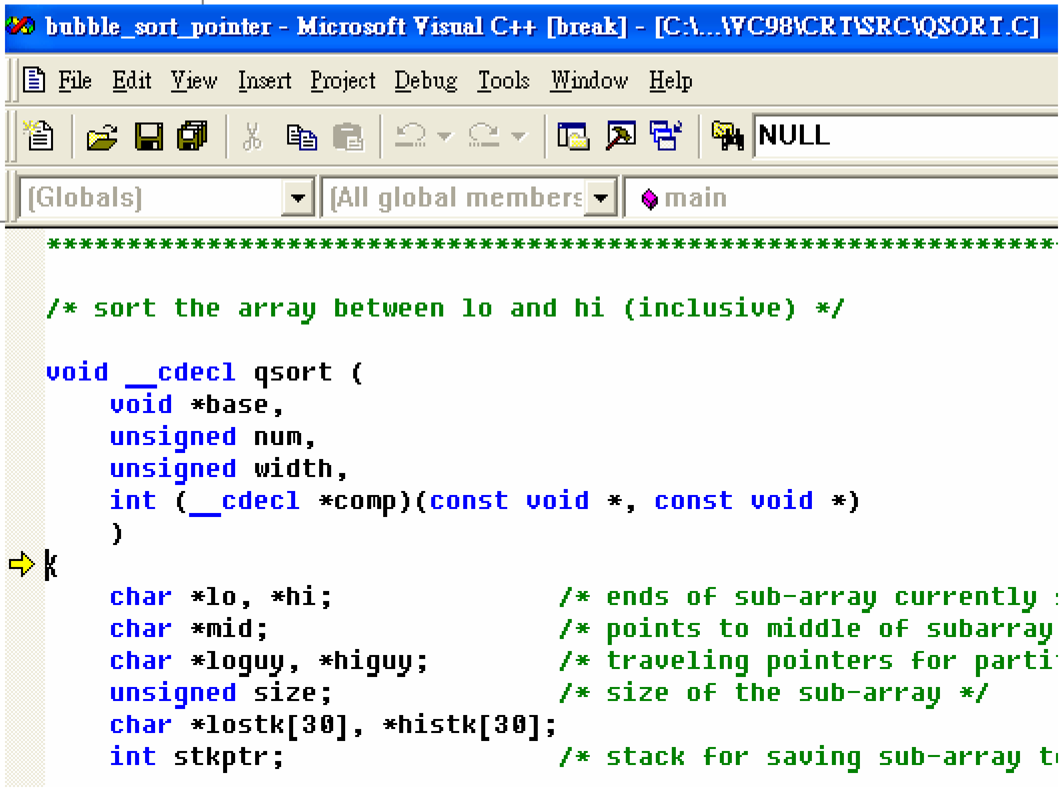
The **qsort** function implements a quick-sort algorithm to sort an array of ***n*** elements, each of ***size*** bytes. The argument ***base*** is a pointer to the base of the array to be sorted. **qsort** overwrites this array with the sorted elements. The argument ***cmp*** is a pointer to a user-supplied routine that compares two array elements and returns a value specifying their relationship. **qsort** calls the compare routine one or more times during the sort, passing pointers to two array elements on each call.

Framework of bubble sort [13]



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

int  double_comp( double *x, double *y )
{
    return (int)(*x - *y) ;
}

int main( int argc, char* argv[] )
{
    int i ;
    double a[] = { 4.0, 3.0, 1.0, 2.0, 8.0, 7.0 } ;

    qsort( (void*) a, (size_t) 6, sizeof(double),

        (int (*)(const void*, const void*))  &double_comp   ) ;

    for(i = 0 ; 5 >= i ; i++){
        printf("%6.2f  ", a[i]);
    }
    printf("\n") ;

    return 0 ;
}
```

C:\Program Files\Microsoft Visual Studio\VC98\CRT\SRC\QSORT.c


bubble_sort_pointer - Microsoft Visual C++ [break] - [C:\...\VC98\CRT\SRC\QSORT.C]
File Edit View Insert Project Debug Tools Window Help
NULL
[Globals]
[All global members
main

```
*****************************************************************

/* sort the array between lo and hi (inclusive) */

void __cdecl qsort (
    void *base,
    unsigned num,
    unsigned width,
    int (__cdecl *comp)(const void *, const void *)
    )
{
    char *lo, *hi;              /* ends of sub-array currently :
    char *mid;                  /* points to middle of subarray
    char *loguy, *higuy;        /* traveling pointers for parti
    unsigned size;              /* size of the sub-array */
    char *lostk[30], *histk[30];
    int stkptr;                 /* stack for saving sub-array t
```

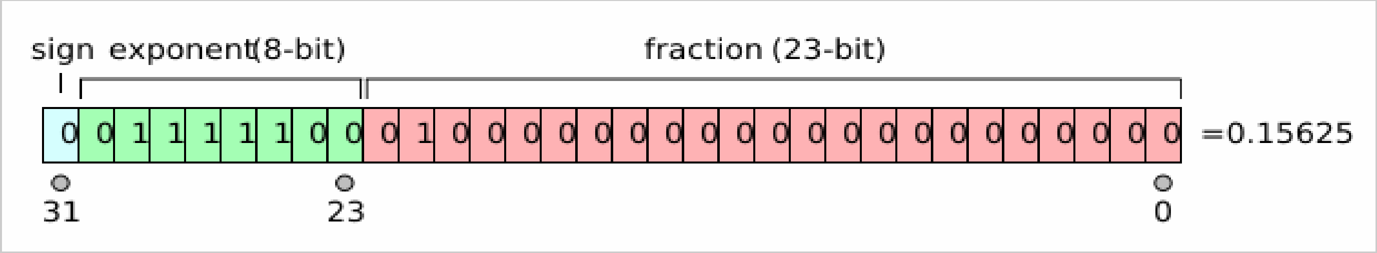
Exercise: you can trace the source code to find out quick-sort algorithm

OutLine

- Memory address and pointer
- Pointer and array
- Call-by-value
- Pointer array: pointers to pointers
- Function pointer
- Application of pointer

Application: How to extract value each field of a floating number

single precision



Bit values for the IEEE 754 32bit float 0.15625

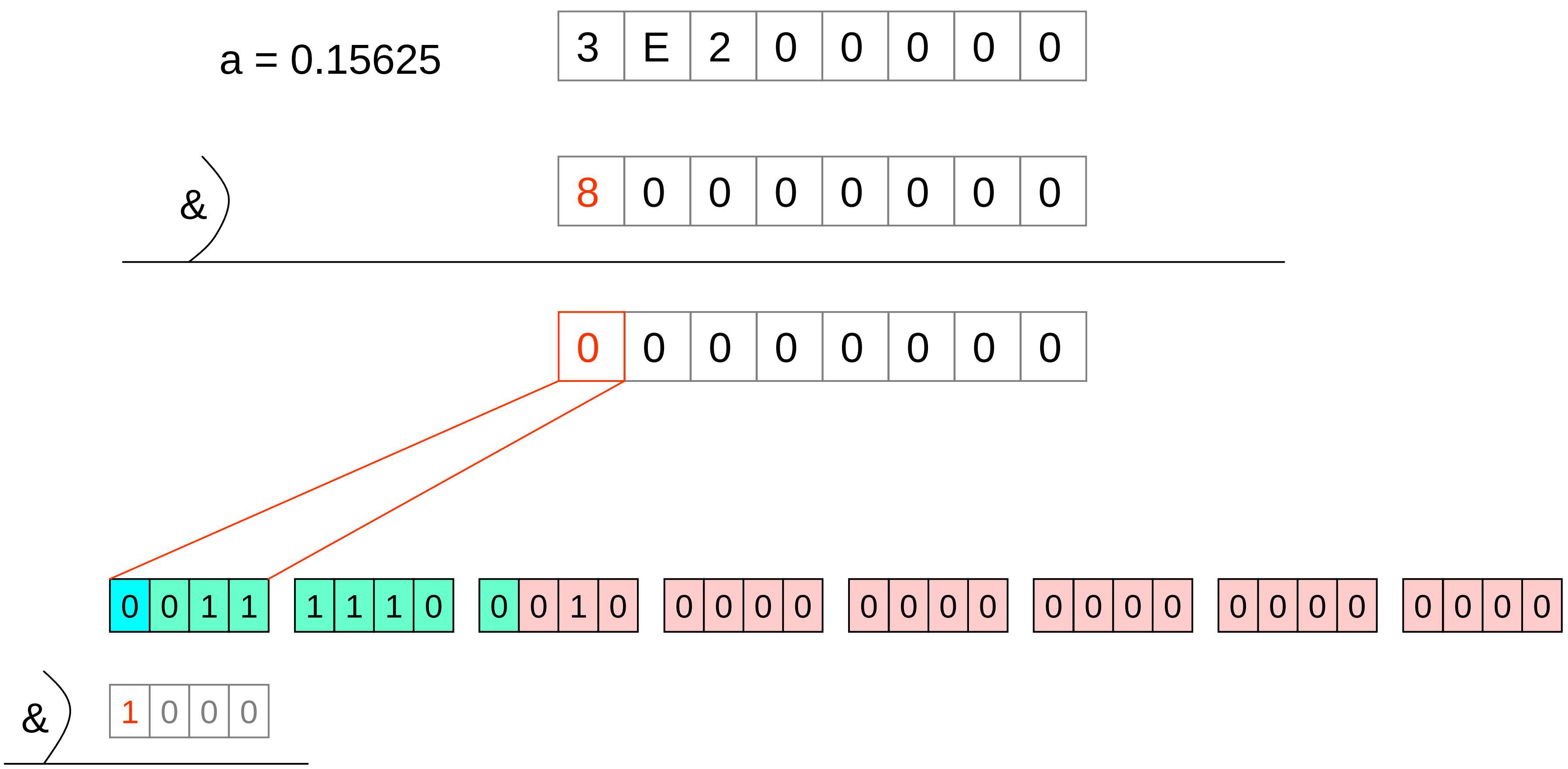
	0 0 1 1	1 1 1 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
16進位	3	E(14)	2	0	0	0	0	0

```
float a = 0.15625 ;
int   *aInt_ptr = (int*) &a ;
```

Question: what is result of

printf (“%x”, **aInt_ptr*)

How to extract sign field



a = 0.15625

3	E	2	0	0	0	0	0

&

8	0	0	0	0	0	0	0

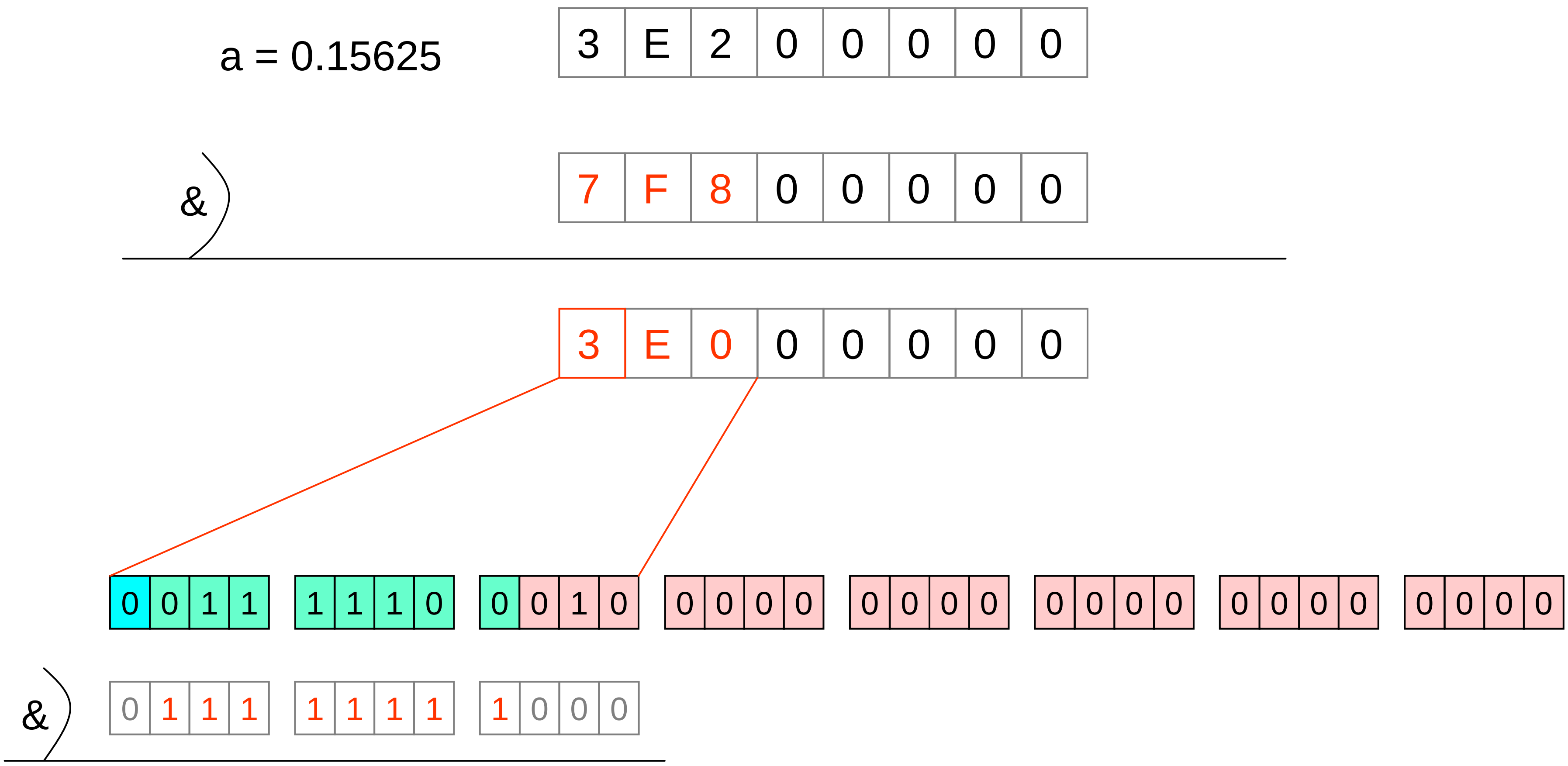
0	0	0	0	0	0	0	0

0 0 1 1   1 1 1 0   0 0 1 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0

&

1 0 0 0

How to extract exponent field



a = 0.15625

3	E	2	0	0	0	0	0

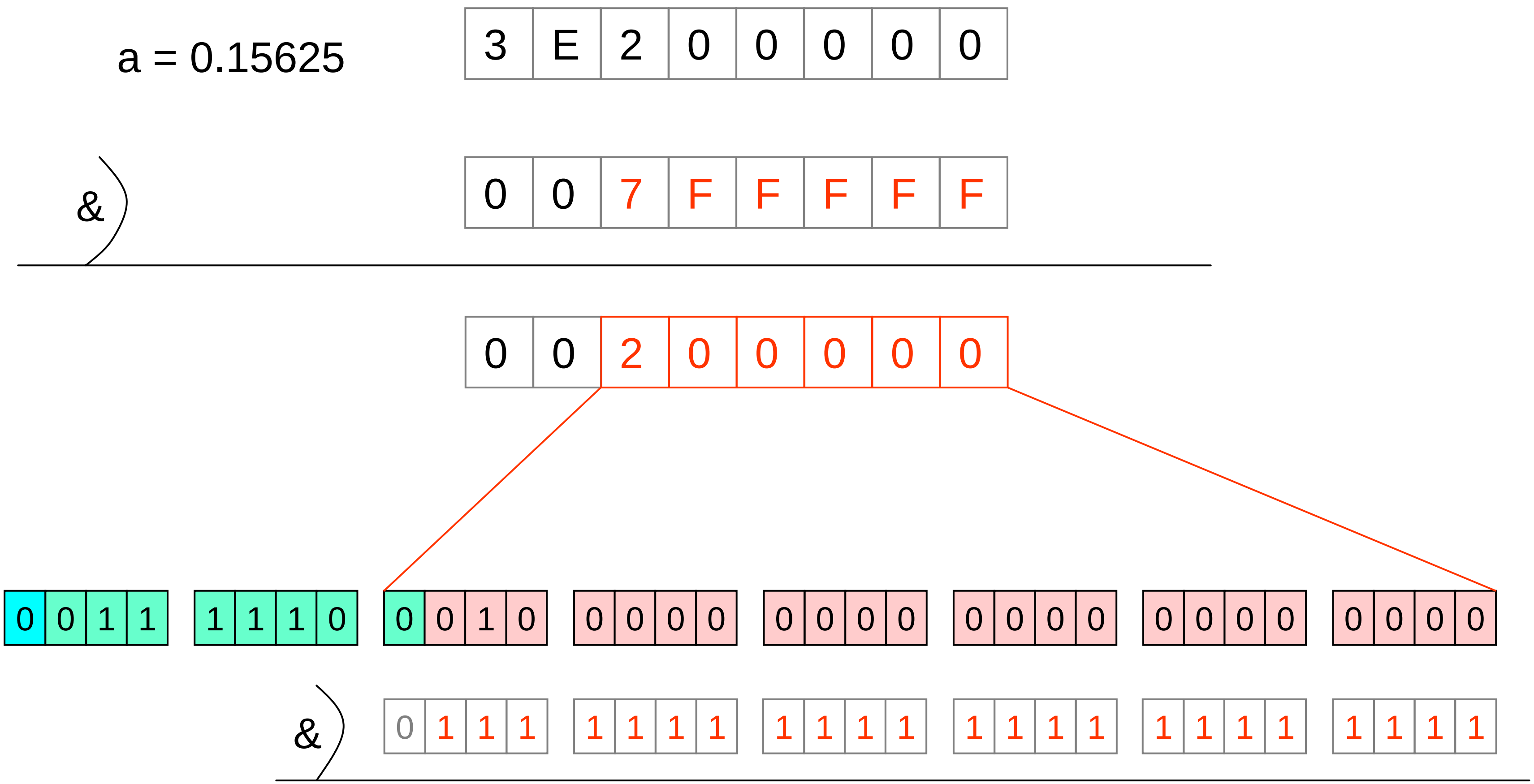
&

7	F	8	0	0	0	0	0

3	E	0	0	0	0	0	0

```
0 0 1 1   1 1 1 0   0 0 1 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 0
&
0 1 1 1   1 1 1 1   1 0 0 0
```

How to extract fraction field



a = 0.15625   3 E 2 0 0 0 0 0

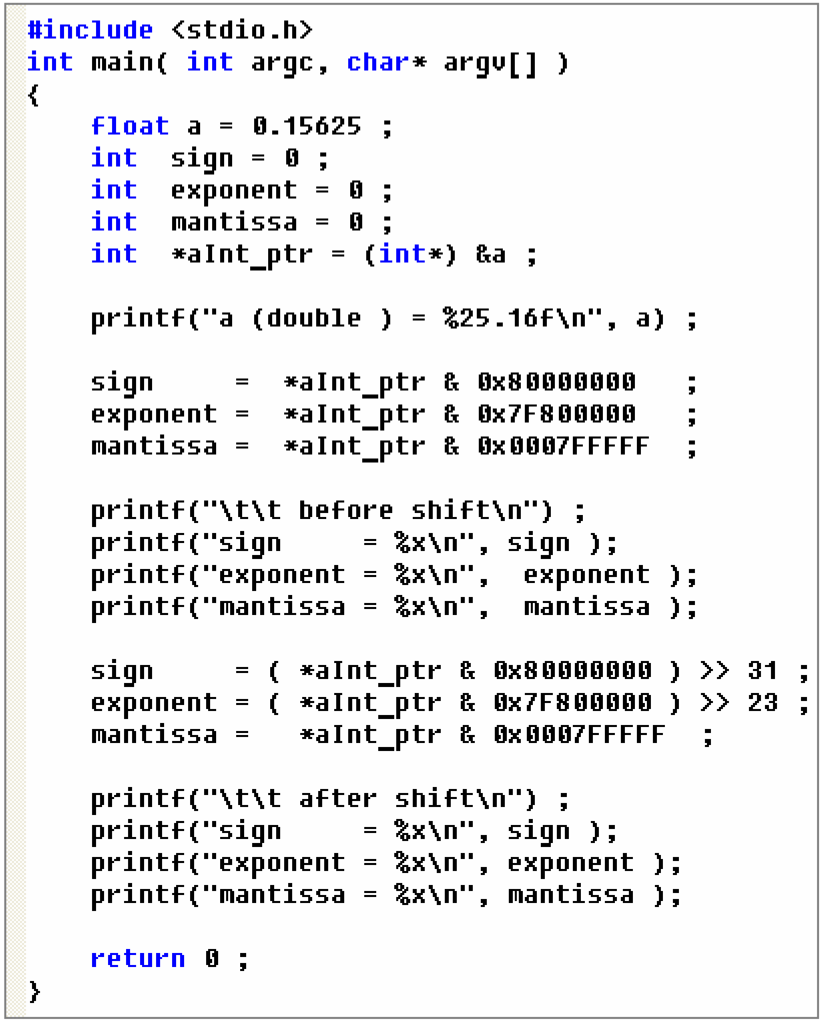
&   0 0 7 F F F F F

0 0 2 0 0 0 0 0

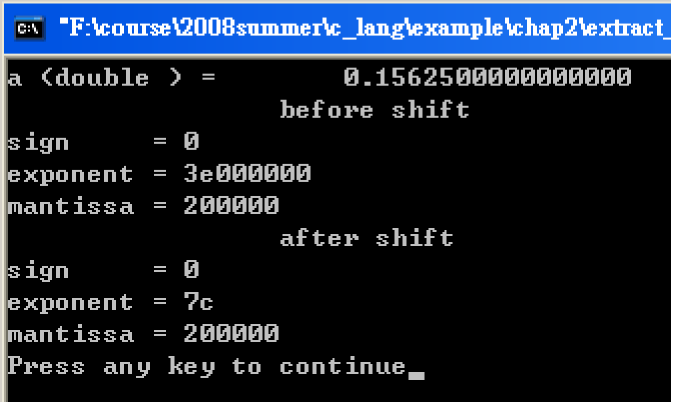
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

&   0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Report value of sign, exponent and fraction fields



```c
#include <stdio.h>
int main( int argc, char* argv[] )
{
    float a = 0.15625 ;
    int  sign = 0 ;
    int  exponent = 0 ;
    int  mantissa = 0 ;
    int  *aInt_ptr = (int*) &a ;

    printf("a (double ) = %25.16f\n", a) ;

    sign     =  *aInt_ptr & 0x80000000   ;
    exponent =  *aInt_ptr & 0x7F800000   ;
    mantissa =  *aInt_ptr & 0x0007FFFFF  ;

    printf("\t\t before shift\n") ;
    printf("sign     = %x\n", sign );
    printf("exponent = %x\n",  exponent );
    printf("mantissa = %x\n",  mantissa );

    sign     = ( *aInt_ptr & 0x80000000 ) >> 31 ;
    exponent = ( *aInt_ptr & 0x7F800000 ) >> 23 ;
    mantissa =   *aInt_ptr & 0x0007FFFFF  ;

    printf("\t\t after shift\n") ;
    printf("sign     = %x\n", sign );
    printf("exponent = %x\n", exponent );
    printf("mantissa = %x\n", mantissa );

    return 0 ;
}
```



```
a (double ) =        0.1562500000000000
                 before shift
sign     = 0
exponent = 3e000000
mantissa = 200000
                 after shift
sign     = 0
exponent = 7c
mantissa = 200000
Press any key to continue_
```

Question: interpret value of sign, exponent and fraction after shift