System Level Programming

Software College of SCU

Instructor: Shu, Li

week08

Unit 6. Memory Layout and Allocation

- 6.1 Several Uses of Memory
- 6.2 Memory-related Bugs
- 6.3 Garbage Collection

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.1 Review of Pointers in C(1/4)

- Here, var occupies 4 bytes, *var_pt also occupies 4 bytes

```
// take the address of a variable
int var;             // declare the variable
int *var_ptr;      // declare the pointer
var_ptr = &var; // take the address of var


*var_ptr = 4;     // stores 4 into var
// access variable pointed to by var_ptr
if (var == *var_ptr)
      printf("ok\n");
```

6.2.1 Review of Pointers in C(2/4)

- Memory Allocation on the Heap

```
// allocate an integer with malloc
// the result must be coerced into an (int *):
var_ptr = (int *) malloc(sizeof(int));
*var_ptr = 4;

// free the memory
free(var_ptr);
var_ptr = NULL;
```

6.2.1 Review of Pointers in C(3/4)

- Pointers and integers can be added
  - pointer +- n //means n elements away

```
double *ptr = x;
   // point to first element in x
   for (int i = 0; i < n; i++)  {
        *ptr++ *= y;
   } //* ++ same priority, so first *ptr *=y then ptr++
```

6.2.1 Review of Pointers in C(4/4)

```
// declare a structure
typedef struct {
        int int_field;
        double dbl_field;
} my_struct_type;
// allocate a structure on the heap
my_struct_type* s;
s = (my_struct_type*) malloc(sizeof(my_struct_type));
// initialize fields of s
s->int_field = 0;
s->dbl_field = 0.0;
// access s in another way:
(*s).int_field = 0;   // equivalent to s->int_field = 0;
```

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.2 Making and Using Bad References(1/10)

- The classic ***scanf*** bug
  - Note that you should supply the address rather than the variable
    - Use &i; instead of i
  - It is important in your exam.

```
int i;
double d;
// wrong!!!
scanf("%d %lf", i, d);
// here is the correct call:
scanf("%d %lf", &i, &d);
```

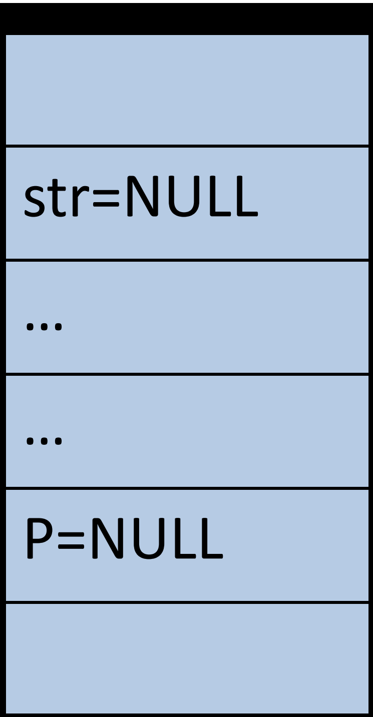
6.2.2 Making and Using Bad References(2/10)

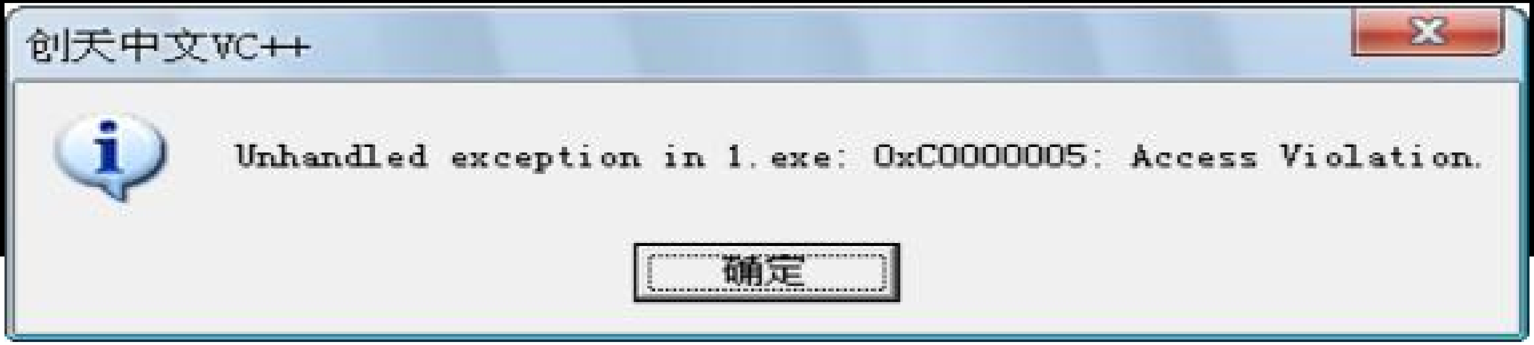
- The pointer is not pointing to the right location or is not properly initialized.
- It might cause the program to crash.
- Here, *p is not pointing to int a[].
- To solve it, p = a;
- Set pointers to NULL if do not know the correct initial value.

```
int sum(int a[], int n)
{
    int* p;
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += *p++;
}
```

6.2.2 Making and Using Bad References(3/10)

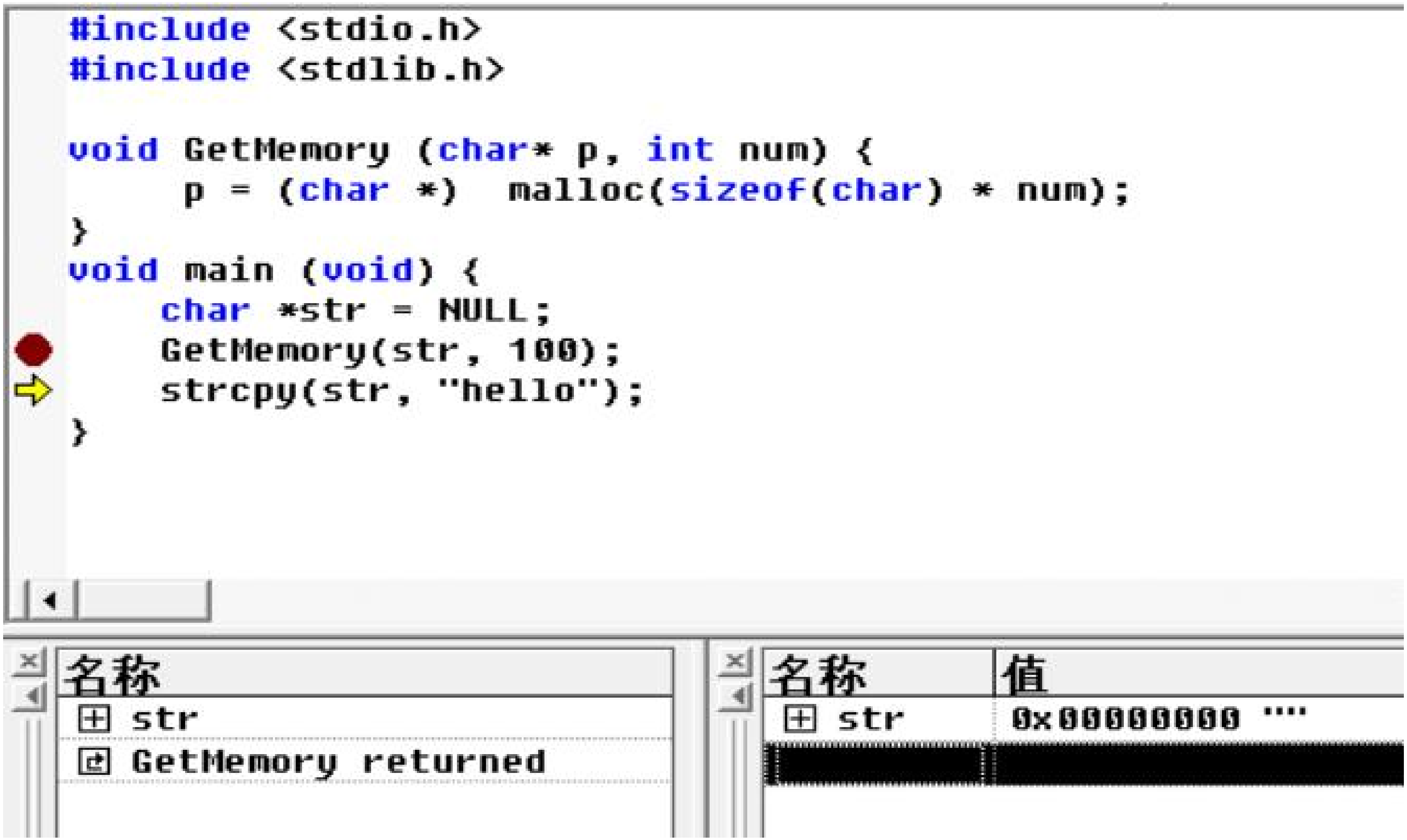
```
void GetMemory (char* p, int num) {
    p = (char *)  malloc(sizeof(char) * num);
}
void main (void) {
    char *str = NULL;
    GetMemory(str, 100);
    strcpy(str, "hello");
}
```





```
Unhandled exception in 1.exe: 0xC0000005: Access Violation.
```

6.2.2 Making and Using Bad References(4/10)



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

void GetMemory (char* p, int num) {
     p = (char *)  malloc(sizeof(char) * num);
}
void main (void) {
    char *str = NULL;
    GetMemory(str, 100);
    strcpy(str, "hello");
}
```

名称
⊞ str
GetMemory returned

名称	值
⊞ str	0x00000000 ""

6.2.2 Making and Using Bad References(5/10)

- Solve the problem: Using Pointer’s Pointer to Correct

```
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
void GetMemory (char** p, int num) {
    *p = (char *)  malloc(sizeof(char) * num);
}
void main (void) {
    char *str = NULL;
    GetMemory(&str, 100);
    strcpy(str, "hello");
    free(str);
}
```

6.2.2 Making and Using Bad References(6/10)

- Multiple indirection (多重引用)

```
int a = 3;
int* b = &a;
int** c = &b;
int*** d = &c;
```

```
*d == c;
**d == *c == b;
***d == **c == *b == a
```

6.2.2 Making and Using Bad References(7/10)

- Solve the problem: Using Return Value to Pass Dynamically Allocated Memory

```
char *GetMemory2(int num) {
    char *p = (char *) malloc(sizeof(char) * num);
    return p;
}
void main (void) {
    char *str = NULL;
    str = GetMemory2(100);
    strcpy(str, "hello");
    free(str);
}
```

6.2.2 Making and Using Bad References(8/10)

```
void release_foo(int * p) {
    free(p);
    p = NULL;
}

int * pointer;
//.... 分配给pointer内存

int bar()  {
    release_foo(pointer);
    return 0;
}
```

Wild Pointer

```
int bar() {
    release_foo(pointer);
   //
    pointer = NULL;
    return 0;
}
```

6.2.2 Making and Using Bad References(9/10)

- Referencing Nonexistent Variables
  - Forgetting that local variables disappear when a function returns
  - Here, it might return any value once the activation record is removed.
  - Never “return” the pointer which points to memory on stack (不要用return语句返回指向“栈内存”的指针)

```
int* ptr = ptr_to_zero() ;

*prt = -1;

//warning in complile

//but on error when execution, both debug
and release version.
```

```
int* ptr_to_zero()  {

        int i = 0;

        return &i;

}
```

6.2.2 Making and Using Bad References(10/10)

- Return Pointer to Stack Memory

```
char *GetString(void){
    char p[ ] = "hello world";
    return p;
    // 编译器将提出警告
}
void main (void) {
    char *str = NULL;
    str = GetString();
    // str 的内容是垃圾
    cout<< str << endl;
}
```

WRONG!!

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.3 Overwriting Memory (1/5)

```
#define array_size 100
int* a = (int *) malloc(sizeof(int) * array_size);
for (int i = 0; i <= array_size; i++)
     a[i] = NULL;
```

- off-by-one errors
  - Here, i will be incremented from 0 to array_size, not array_size – 1;
  - The solution is: i < array_size not i <= array_size

6.2.3 Overwriting Memory (2/5)

```
#define array_size 100
int *a = (int *) malloc(array_size);
a[99] = 0; // this overwrites memory beyond the block
```

- Allocating the (possibly) wrong sized object
  - Here, the memory allocated is 100 bytes, not 400 bytes and a[] is defined as array pointer
  - The solution is:

```
int *a = (int *) malloc( array_size* sizeof(int));
```

6.2.3 Overwriting Memory (3/5)

```
char s[8];
int i;
gets(s);  /* reads “123456789” from stdin */
```

- Not checking the max string size
- Basis for classic buffer overflow attacks
  - 1988 Internet worm
  - Modern attacks on Web servers

6.2.3 Overwriting Memory (4/5)

```
char *heapify_string(char *s)  {
    int len = strlen(s);
    char *new_s = (char *) malloc(len);
    strcpy(new_s, s);
    return new_s;
}
```

- String must be terminated by 0x00;
- The solution is:

```
char *new_s = (char *) malloc(len + 1);
```

6.2.3 Overwriting Memory (5/5)

```
// decrement a if a is greater than zero:
void dec_positive(int* a)
{
        *a--; // decrement the integer:  (*a)--
        if (*a < 0) *a = 0; // make sure a is positive
}
```

- Operator precedence
  - Note that *a-- will decrement the pointer not the value.
  - We should use (*a)-- to refer to the decrement of value

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.4 Twice free(1/2)

```
void my_write(int* x)  {
          ... use x ...
     free(x);
}
int* x = (int*) malloc(sizeof(int) * N);       ...
my_read(x);       ...
my_write(x);
free(x); //oops, x is freed in my_write()!
```

- It means the memory pointer was freed twice.
- Here, it free the memory in the routine my_write(), but it is freed in the main

6.2.4 Twice free(2/2)

- Referencing Freed Blocks, Evil!

```
x = malloc(N*sizeof(int));
  <manipulate x>
free(x);
   ...
y = malloc(M*sizeof(int));
for (i=0; i<M; i++)
   y[i] = x[i]++;
```

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.5 Memory Leaks(1/4)

- The failure to deallocate (free) a block of memory when it is no longer needed is often called a **memory leak**.
  - The result is that the system will run out of memory.
  - Programs that run for long periods of time must be careful to de-allocate memory when it becomes free.
    - For instance: Operating systems and Web servers

Slow, long-term memory killer!

6.2.5 Memory Leaks(2/4)

- Example of Memory Leak

```
foo() {
   int* x = malloc ( N * sizeof (int) );
   ...
   return;
}
```

6.2.5 Memory Leaks(3/4)

```
void my_function(char *msg) {
   // allocate space for a string
   char *full_msg = (char *) malloc(strlen(msg) + 100);
   strcpy(full_msg, “ error was encountered: ");
   strcat(full_msg, msg);
   if (!display(full_msg)) return;
   free(full_msg);
}
```

- If it is true in the (full_msg), it will not execute to free(full_msg)
- The memory leak occurs by error or unusual returns skipping the code that was intended to free memory.

6.2.5 Memory Leaks(4/4)

```
typedef struct {
            char *name;
            int age;
            char *address;
            int phone;
} Person;
void my_function() {
            Person *p = (Person *) malloc(sizeof(Person));
            p->name = (char *) malloc(M); ...
            p->address = (char *) malloc(N); ...
            free(p); // what about name and address?
}
```

- Freeing only part of a data structure.
  - A Person structure was allocated and freed, but the fields of the Person structure, name and address, were allocated but not freed.
  - **How to fix it??**

6.2 Memory-related Bugs

- //memory use and related errors
- 6.2.1 Review of Pointers in C
- 6.2.2 Making and Using Bad References
- 6.2.3 Overwriting Memory
- 6.2.4 Twice free
- 6.2.5 Memory Leaks

- //tools to detect related errors
- **6.2.6 Exterminating Memory Bugs**

6.2.6 Exterminating Memory Bugs

- 6.2.6.1 Introduction
- 6.2.6.2 Debugging_Malloc Lab
- 6.2.6.3 Debug Heap Management with VC

6.2.6.1 Introduction(1/4)

- Memory related errors:
  - 1/Memory leak
  - 2/Overwriting Memory
  - 3/Abuse dangle/wild pointer

6.2.6.1 Introduction(2/4)

- In our last lab, we try to detect the following memory related errors by adding extra information.

2

  - Error #1: Writing past the beginning of the user's block (through the fence)
  - Error #2: Writing past the end of the user's block (through the fence)
  - Error #3: Corrupting the header information

3

  - Error #4: Attempting to free an unallocated or already-freed block

1

  - Error #5: Memory leak detection (user can use ALLOCATEDSIZE to check for leaks at the end of the program)

6.2.6.1 Introduction(3/4)

- The extra information may include:
  - The file name and line number where the allocation occurred.
  - The status: whether the block is allocated or free.
  - Padding(填充) before and after the block.
    - This padding is filled with a known value such as "0xdeadbeef" so that if memory is overwritten near the boundaries of the block, the changes to the known values can easily be detected.
  - Links to other blocks.
    - These enable allocated blocks to be scanned and checked.
  - An allocation sequence number.
    - This helps to identify blocks.

- **How to detect accessing by dangle/wild pointer?**
- **What kind of extra information can help?**

6.2.6.1 Introduction(4/4)

- Basic approach of detect memory bugs
  - record extra information about blocks whenever a block is allocated
  - check that information for consistency whenever a block is freed.

6.2.6 Exterminating Memory Bugs

- 6.2.6.1 Introduction
- 6.2.6.2 Debugging_Malloc Lab
- 6.2.6.3 Debug Heap Management with VC

6.2.6.2 Debugging_Malloc Lab(1/8)

- Wrapper around conventional malloc

- Detects memory bugs when deallocating(extra infor. added in malloc)
  - memory overwrites that corrupt heap structures
  - some instances freeing blocks multiple times
  - memory leaks

- Cannot detect all memroy bugs
  - Overwrites into the middle of allocated blocks
  - Referencing freed blocks

6.2.6.2 Debugging_Malloc Lab(2/8)

```
#define malloc(size) my_malloc(size, __FILE__, __LINE__)
#define free(p) my_free(p, __FILE__, __LINE__)
```

```
#ifdef DEBUG
#include <my_malloc.h>
#endif
#define DEBUG
main() {
   ...
   p = malloc(128);
   ...
   free(p);
   ...
}
```

6.2.6.2 Debugging_Malloc Lab(3/8)

- Predefined Macros
  - __FILE__
    - The name of the current source file.
  - __LINE__
    - The line number in the current source file.

- __FILE and __LINE__ 编译器定义的宏
  - 编译器会把__FILE__替换成“当前在编译的文件的路径”名字符串；编译器会把__LINE__替换成当前这行代码行号这个数字。
  - VS中需要 define _CRTDBG_MAP_ALLOC ; 并开启内存检测，如调用_CrtDumpMemoryLeaks();否则_FILE_,_LINE_的内容为0，0

6.2.6.2 Debugging_Malloc Lab(4/8)

- User-defined Macros
  - **#define identifiers expressions**
  - Object-like macros take no argument
  - Function-like macros take argument(s)

```
// my_malloc.h
#define malloc(size) my_malloc(size, __FILE__, __LINE__)
#define free(p) my_free(p, __FILE__, __LINE__)
```

6.2.6.2 Debugging_Malloc Lab(5/8)

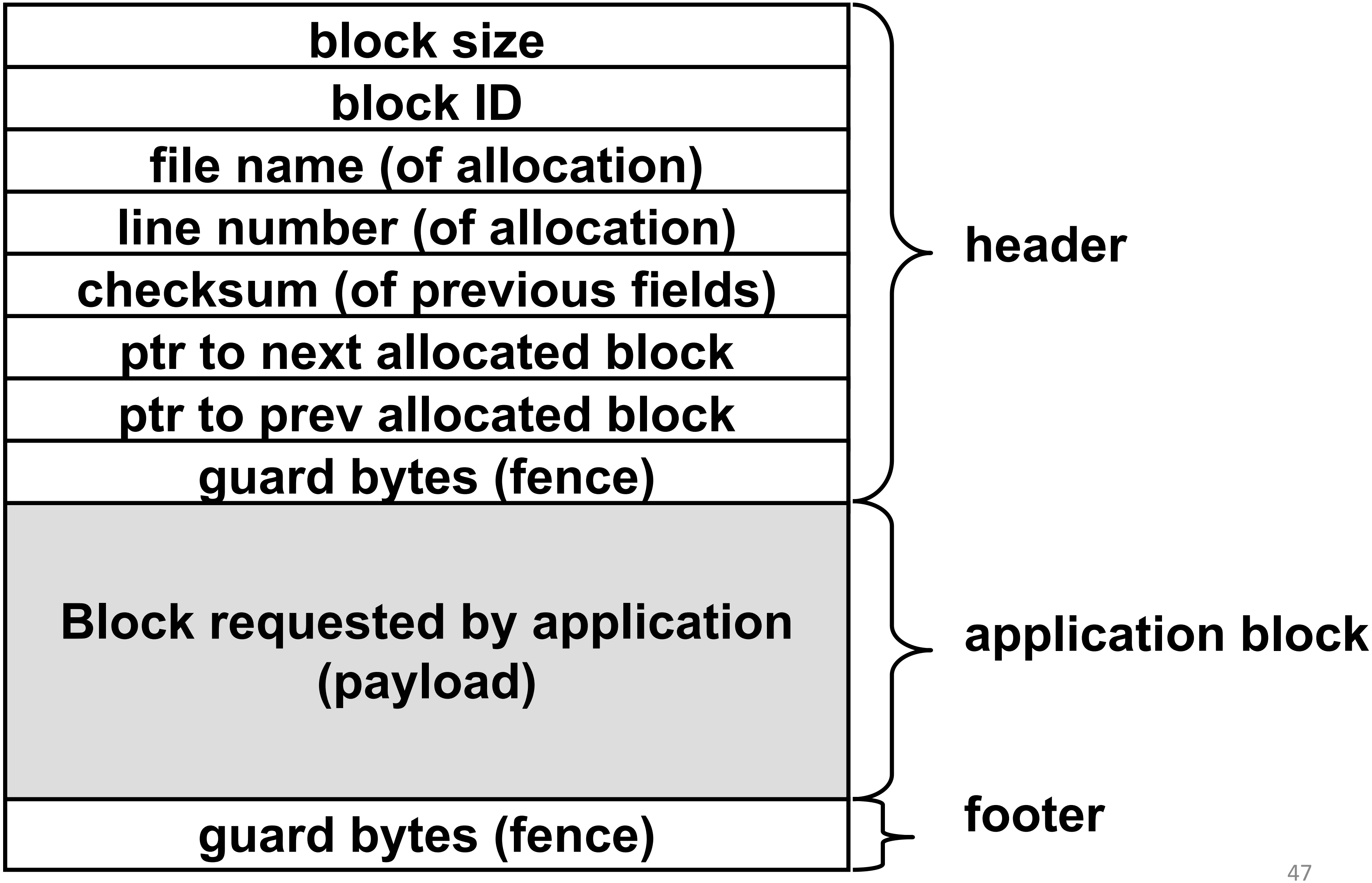
- Conditional-compilation directive

```
#ifdef DEBUG
#include <my_malloc.h>
#endif

#define DEBUG
main() {
   …
   p = malloc(128);
   …
   free(p);
   …
}
```

6.2.6.2 Debugging_Malloc Lab(6/8)

```
void *my_malloc(int size, char *file, int line) {
  <prologue code> // 前期代码
  p = malloc(...);
  <epilogue code>  // 后期代码
  return q;
}

void my_free(void *q, char *file, int line) {
  <prologue code>
  free(p);
  <epilogue code>
}
```

6.2.6.2 Debugging_Malloc Lab(7/8)



block size

block ID

file name (of allocation)

line number (of allocation)

checksum (of previous fields)

ptr to next allocated block

ptr to prev allocated block

guard bytes (fence)

Block requested by application (payload)

guard bytes (fence)

header

application block

footer

6.2.6.2 Debugging_Malloc Lab(8/8)

- my_malloc(size):
  - p = malloc(size + sizeof(header) + sizeof(footer));
  - add p to list of allocated blocks
  - initialize fence to 0xdeadbeef
  - return pointer to application block
- my_free(p):
  - already free (line # = 0xfeeefeeefeeefeee)?
  - checksum OK?
  - guard bytes OK?
  - line # = 0xfefefefefefefefe;
  - free(p - sizeof(hdr));
- Leaking ?

6.2.6 Exterminating Memory Bugs

- 6.2.6.1 Introduction
- 6.2.6.2 Debugging_Malloc Lab
- 6.2.6.3 Debug Heap Management with VC

6.2.6.3 Debug Heap Management with VC

- Inside CRT(C RunTime Lib) : Debug Heap Management with Visual Studio
  - 1. Memory Management Function (Review)
  - 2. 0xCDs, 0xDDs and 0xFDs in memory
  - 3. _CrtMemBlockHeader
  - 4. _crtDbgFlag

1. Memory Management Function (Review)(1/5)

- ***c/c++***
  - ***malloc***
    - The implementation of the C++ operator ***new*** is based on malloc.
  - ***free***
    - The implementation of C++ operator *delete* is based on free.
- Win32 API
  - ***HeapAlloc***
    - The allocated memory is not movable.
  - ***HeapFree***
    - Free blocks allocated from a heap by the HeapAlloc or HeapReAlloc function.

1. Memory Management Function (Review)(2/5)

- ***_heap_alloc_dbg*** (VS)
  - void *_malloc_dbg(size_t size, int blockType, const char *filename, int linenumber); 最终调用下图函数
  - Allocates a block of memory in the heap with additional space for a debugging header and overwrite buffers (debug version only).

```
void * __cdecl _heap_alloc_dbg(
        size_t nSize,
        int nBlockUse,
        const char * szFileName,
        int nLine
        )
```

1. Memory Management Function (Review)(3/5)

- ***_free_dbg*** (VS)
  - void _free_dbg(void *userData, int blockType);
  - Debug version of free; performs a validity check on all specified files and block locations
  - Both free and _free_dbg free a memory block in the base heap, but _free_dbg accomodates two debugging features:
    - blockType(CLIENT_BLOCK/_NORMAL_BLOCK/ _IGNORE_BLOCK)
    - the ability to keep freed blocks in the heap's linked list to simulate low memory conditions and a block type parameter to free specific allocation types.

1. Memory Management Function (Review)(4/5)

- normal block（普通块）： 由你的程序分配的内存
- client block（客户块）：特殊类型的内存块，MFC分配的全部属于该类型。MFC new 操作视具体情况既可以为所创建的对象建立普通块，也可以为之建立客户块。
- CRT block（CRT 块）：是由 C RunTime Library 供自己使用而分配的内存块。由 CRT 库自己来管理这些内存的分配与释放，我们一般不会在内存泄漏报告中发现 CRT 内存泄漏，除非程序发生了严重的错误（例如 CRT 库崩溃）。

1. Memory Management Function (Review)(5/5)

- 除了上述的类型外，还有下面这两种类型的内存块，它们不会出现在内存泄漏报告中：
  - free block（空闲块）：已经被释放(free)的内存块。（可以选择不释放xDD，以检测是否还在非法使用）
  - Ignore block（忽略块）：这是程序员显式声明过不要在内存泄漏报告中出现的内存块。

6.2.6.3 Debug Heap Management with VC

- Inside CRT : Debug Heap Management with Visual C++
  - 1. Memory Management Function (Review)
  - 2. 0xCDs, 0xDDs and 0xFDs in memory
  - 3. _CrtMemBlockHeader
  - 4. _crtDbgFlag

2. 0xCDs, 0xDDs and 0xFDs in memory(1/4)

0xCD	**C**lean Memory	Allocated memory via malloc or new but never written by the application.
0xDD	**D**ead Memory	Memory that has been released with delete or free. It is used to detect writing through dangling pointers.
0xFD	**F**ence	Memory Also known as "no mans land." This is used to wrap the allocated memory (like surrounding it with fences) and is used to detect indexing arrays out of bounds.

How to detect accessing by dangle/wild pointer?
What kind of extra information can help?

2. 0xCDs, 0xDDs and 0xFDs in memory(2/4)

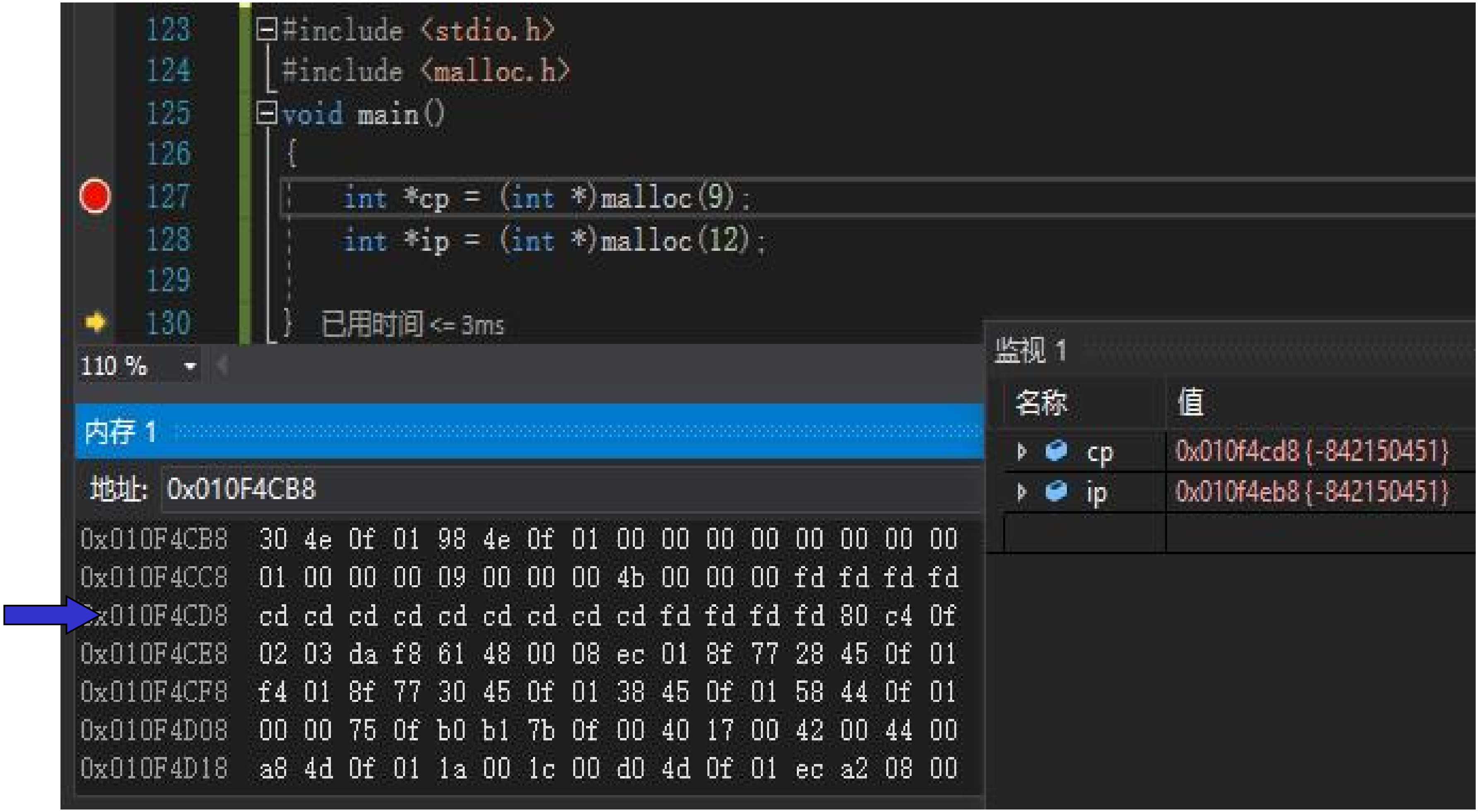
```
// DBGHEAP.C
static unsigned char _bNoMansLandFill = 0xFD;
/* fill no-man's land with this */

static unsigned char _bDeadLandFill = 0xDD;
/* fill free objects with this */

static unsigned char _bCleanLandFill = 0xCD;
/* fill new objects with this */
```

2. 0xCDs, 0xDDs and 0xFDs in memory(3/4)

```
#include <stdlib.h>
#include <malloc.h>
int main(int argc, char* argv[]) {
        char* buffer = (char*)malloc(12);
    free(buffer);
    buffer = NULL;
    return 0;
}
```

2. 0xCDs, 0xDDs and 0xFDs in memory(4/4)

- Examine Heap Memory



```
#include <stdio.h>
#include <malloc.h>
void main()
{
    int *cp = (int *)malloc(9);
    int *ip = (int *)malloc(12);

}  已用时间 <= 3ms
```

内存 1

地址: 0x010F4CB8

```
0x010F4CB8  30 4e 0f 01 98 4e 0f 01 00 00 00 00 00 00 00 00
0x010F4CC8  01 00 00 00 09 00 00 00 4b 00 00 00 fd fd fd fd
0x010F4CD8  cd cd cd cd cd cd cd cd cd fd fd fd fd 80 c4 0f
0x010F4CE8  02 03 da f8 61 48 00 08 ec 01 8f 77 28 45 0f 01
0x010F4CF8  f4 01 8f 77 30 45 0f 01 38 45 0f 01 58 44 0f 01
0x010F4D08  00 00 75 0f b0 b1 7b 0f 00 40 17 00 42 00 44 00
0x010F4D18  a8 4d 0f 01 1a 00 1c 00 d0 4d 0f 01 ec a2 08 00
```

监视 1

名称	值
cp	0x010f4cd8 {-842150451}
ip	0x010f4eb8 {-842150451}

6.2.6.3 Debug Heap Management with VC

- Inside CRT : Debug Heap Management with Visual C++
  - 1. Memory Management Function (Review)
  - 2. 0xCDs, 0xDDs and 0xFDs in memory
  - 3. _CrtMemBlockHeader
  - 4. _crtDbgFlag

3. _CrtMemBlockHeader(1/14)

- For each allocated block, the CRT keeps information in a structure called _CrtMemBlockHeader, which is declared in DBGINT.H

- For the previous example, 12 bytes are dynamically allocated, but the CRT allocates more than that by wrapping the allocated block with bookkeeping information.

3. _CrtMemBlockHeader(2/14)

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
    struct _CrtMemBlockHeader * pBlockHeaderNext;
    struct _CrtMemBlockHeader * pBlockHeaderPrev;
    char * szFileName;
    int nLine;
    size_t nDataSize;
    int nBlockUse;
    long lRequest;
    unsigned char gap[nNoMansLandSize];
    /* followed by:
     * unsigned char data[nDataSize];
     * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

3. _CrtMemBlockHeader(3/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
     struct _CrtMemBlockHeader * pBlockHeaderNext;
     struct _CrtMemBlockHeader * pBlockHeaderPrev;
     char * szFileName;
     int nLine;

     * unsigned char data[nDataSize];
     * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

pBlockHeaderNext: A pointer to the block allocated just before this one

3. _CrtMemBlockHeader(4/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
    struct _CrtMemBlockHeader * pBlockHeaderNext;
    struct _CrtMemBlockHeader * pBlockHeaderPrev;
    char * szFileName;
```

pBlockHeaderPrev: A pointer to the block that was allocated after the current block.

```
    * followed by:
    * unsigned char data[nDataSize];
    * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

3. _CrtMemBlockHeader(5/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
    struct _CrtMemBlockHeader * pBlockHeaderNext;
    struct _CrtMemBlockHeader * pBlockHeaderPrev;
    char * szFileName;
    int nLine;
```

szFileName: A pointer to the name of the file in which the call to malloc was made, if known.

nLine: The line in the source file indicated by szFileName at which the call to malloc was made, if known.

```
    • unsigned char anotherGap[nNoMansLandSize]; /
} _CrtMemBlockHeader;
```

3. _CrtMemBlockHeader(6/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
    struct _CrtMemBlockHeader * pBlockHeaderNext;
    struct _CrtMemBlockHeader * pBlockHeaderPrev;
    char * szFileName;
    int nLine;
    size_t nDataSize;
    .* unsigned char data[nDataSize];
    .* unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

nDataSize: Number of bytes requested

3. _CrtMemBlockHeader(7/14）

nBlockUse
0 - Freed block, but not released back to the Win32 heap
1 - Normal block (allocated with new/malloc)
2 - CRT blocks, allocated by CRT for its own use

```
    size_t nDataSize;
    int nBlockUse;
    long lRequest;
    unsigned char gap[nNoMansLandSize];
    /* followed by:
     * unsigned char data[nDataSize];
     * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

3. _CrtMemBlockHeader(8/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
     size_t nDataSize;
     int nBlockUse;
     long lRequest;
     unsigned char gap[nNoMansLandSize];
     /* followed by:
      * unsigned char data[nDataSize];
      * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

lRequest: Counter incremented with each Allocation.

3. _CrtMemBlockHeader(9/14）

```
#define nNoMansLandSize 4
typedef struct _CrtMemBlockHeader {
        int nBlockUse;
        long lRequest;
        unsigned char gap[nNoMansLandSize];
        /* followed by:
         * unsigned char data[nDataSize];
         * unsigned char anotherGap[nNoMansLandSize]; */
} _CrtMemBlockHeader;
```

gap: A zone of 4 bytes (in the current implementation) filled with 0xFD, fencing the data block, of nDataSize bytes. Another block filled with 0xFD of the same size follows the data.

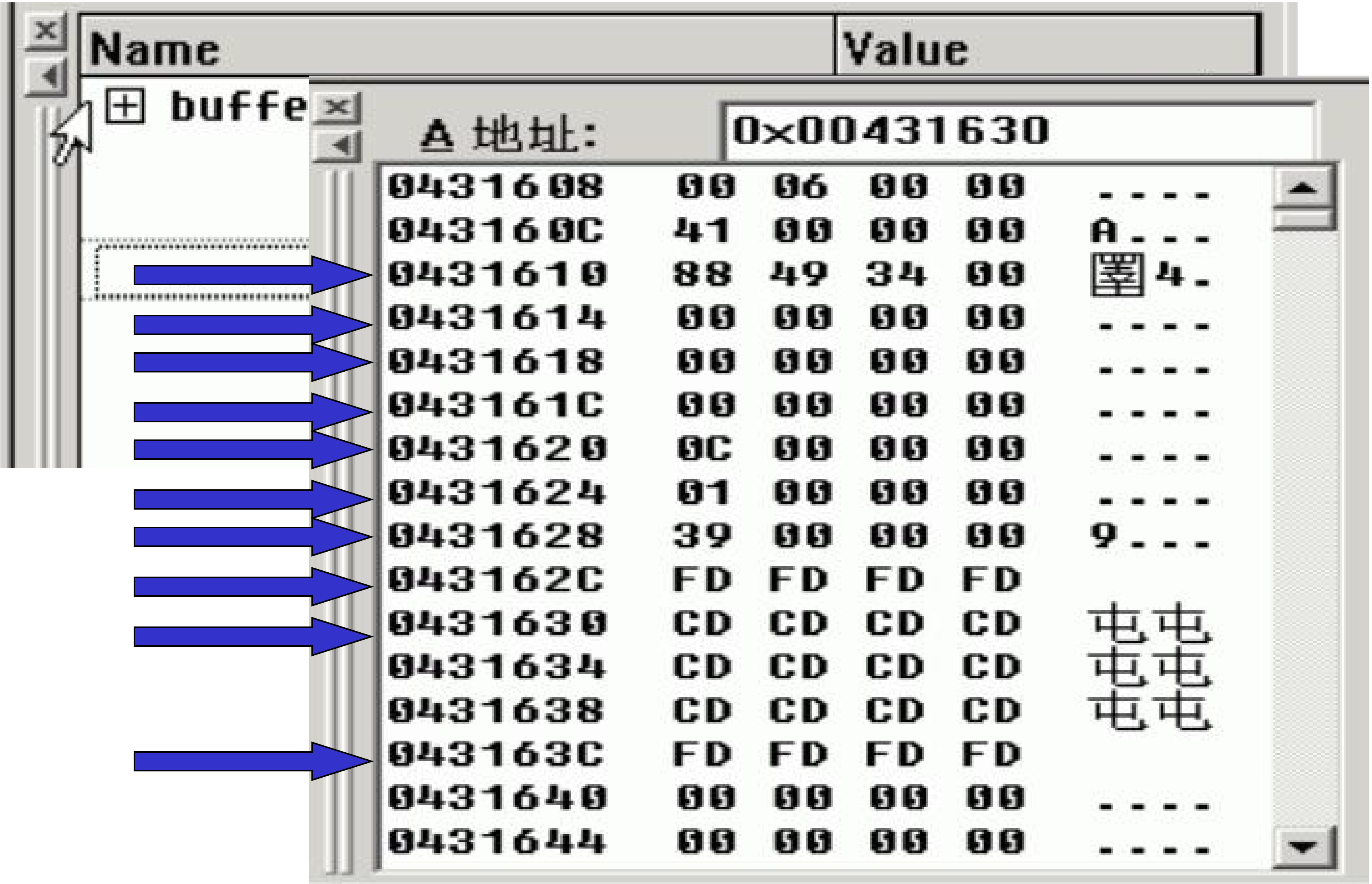
3. _CrtMemBlockHeader(10/14）

- When request 12 bytes on the heap
  - Request 36 more bytes

```
blockSize =
sizeof(_CrtMemBlockHeader) + nSize + nNoMansLandSize;
```

  - Fill the _CrtMemBlockHeader block with bookkeeping information.
  - Initialize the data block with 0xCD and no mans land with 0xFD.

3. _CrtMemBlockHeader(11/14）



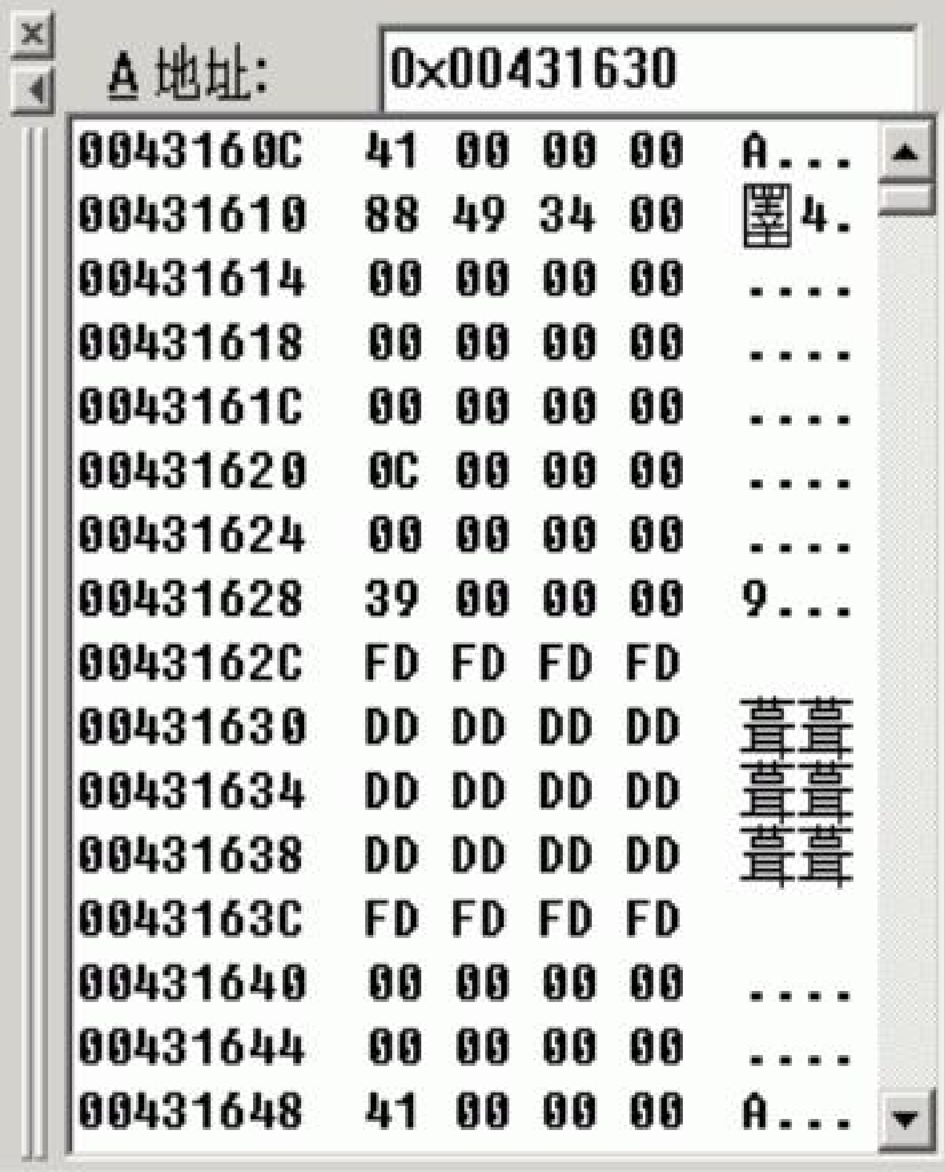
```
0x00431630
```

```
00431608   00  06  00  00   ....
0043160C   41  00  00  00   A...
00431610   88  49  34  00   圛4.
00431614   00  00  00  00   ....
00431618   00  00  00  00   ....
0043161C   00  00  00  00   ....
00431620   0C  00  00  00   ....
00431624   01  00  00  00   ....
00431628   39  00  00  00   9...
0043162C   FD  FD  FD  FD
00431630   CD  CD  CD  CD   屯屯
00431634   CD  CD  CD  CD   屯屯
00431638   CD  CD  CD  CD   屯屯
0043163C   FD  FD  FD  FD
00431640   00  00  00  00   ....
00431644   00  00  00  00   ....
```

3. _CrtMemBlockHeader(12/14)

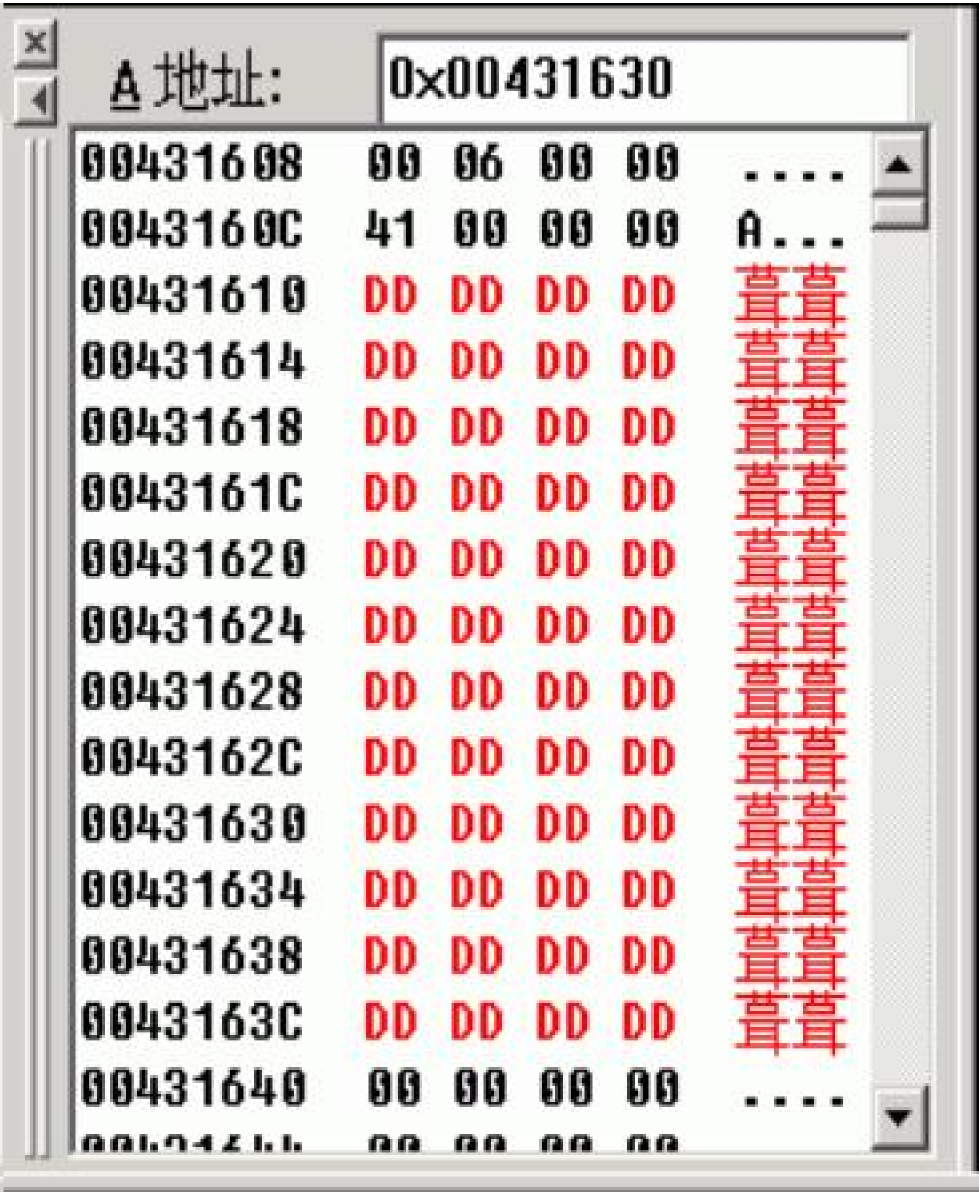
- Free Memory
  - When you call free, the CRT will set the block it requested to 0xDD, indicating this is a free zone.

3. _CrtMemBlockHeader(13/14)



A 地址: 0x00431630

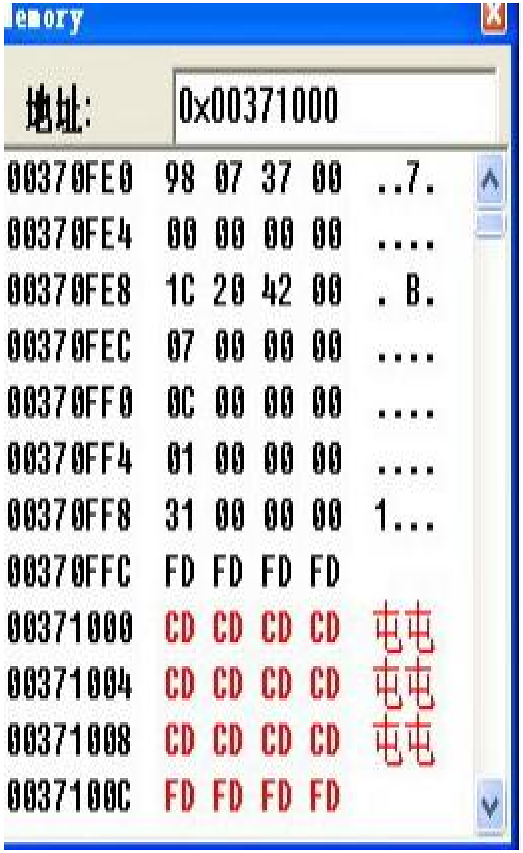
```
0043160C   41  00  00  00   A...
00431610   88  49  34  00   圛4.
00431614   00  00  00  00   ....
00431618   00  00  00  00   ....
0043161C   00  00  00  00   ....
00431620   0C  00  00  00   ....
00431624   00  00  00  00   ....
00431628   39  00  00  00   9...
0043162C   FD  FD  FD  FD
00431630   DD  DD  DD  DD   葺葺
00431634   DD  DD  DD  DD   葺葺
00431638   DD  DD  DD  DD   葺葺
0043163C   FD  FD  FD  FD
00431640   00  00  00  00   ....
00431644   00  00  00  00   ....
00431648   41  00  00  00   A...
```



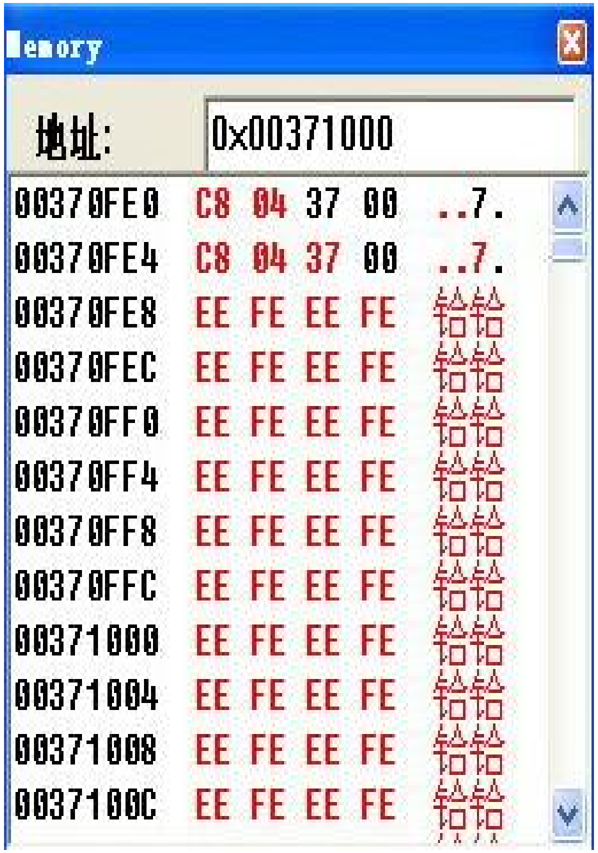
A 地址: 0x00431630

```
00431608   00  06  00  00   ....
0043160C   41  00  00  00   A...
00431610   DD  DD  DD  DD   葺葺
00431614   DD  DD  DD  DD   葺葺
00431618   DD  DD  DD  DD   葺葺
0043161C   DD  DD  DD  DD   葺葺
00431620   DD  DD  DD  DD   葺葺
00431624   DD  DD  DD  DD   葺葺
00431628   DD  DD  DD  DD   葺葺
0043162C   DD  DD  DD  DD   葺葺
00431630   DD  DD  DD  DD   葺葺
00431634   DD  DD  DD  DD   葺葺
00431638   DD  DD  DD  DD   葺葺
0043163C   DD  DD  DD  DD   葺葺
00431640   00  00  00  00   ....
00431644   00  00  00  00
```

3. _CrtMemBlockHeader(14/14)

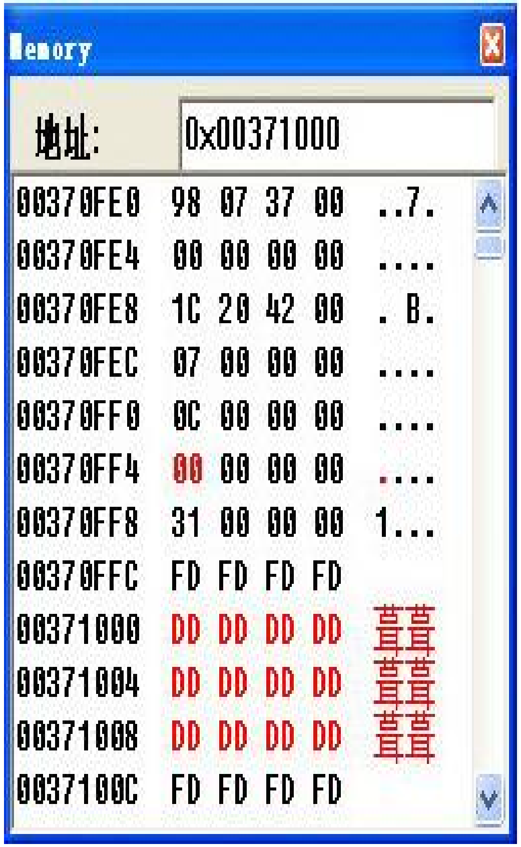


```
地址: 0x00371000
00370FE0  98 07 37 00  ..7.
00370FE4  00 00 00 00  ....
00370FE8  1C 20 42 00  . B.
00370FEC  07 00 00 00  ....
00370FF0  0C 00 00 00  ....
00370FF4  01 00 00 00  ....
00370FF8  31 00 00 00  1...
00370FFC  FD FD FD FD
00371000  CD CD CD CD  屯屯
00371004  CD CD CD CD  屯屯
00371008  CD CD CD CD  屯屯
0037100C  FD FD FD FD
```



```
地址: 0x00371000
00370FE0  C8 04 37 00  ..7.
00370FE4  C8 04 37 00  ..7.
00370FE8  EE FE EE FE  铪铪
00370FEC  EE FE EE FE  铪铪
00370FF0  EE FE EE FE  铪铪
00370FF4  EE FE EE FE  铪铪
00370FF8  EE FE EE FE  铪铪
00370FFC  EE FE EE FE  铪铪
00371000  EE FE EE FE  铪铪
00371004  EE FE EE FE  铪铪
00371008  EE FE EE FE  铪铪
0037100C  EE FE EE FE  铪铪
```

_CRTDBG_DELAY_FREE_MEM_DF is NOT set



```
地址: 0x00371000
00370FE0  98 07 37 00  ..7.
00370FE4  00 00 00 00  ....
00370FE8  1C 20 42 00  . B.
00370FEC  07 00 00 00  ....
00370FF0  0C 00 00 00  ....
00370FF4  00 00 00 00  ....
00370FF8  31 00 00 00  1...
00370FFC  FD FD FD FD
00371000  DD DD DD DD  葺葺
00371004  DD DD DD DD  葺葺
00371008  DD DD DD DD  葺葺
0037100C  FD FD FD FD
```

_CRTDBG_DELAY_FREE_MEM_DF is set

6.2.6.4 Debug Heap Management with VC

- Inside CRT : Debug Heap Management with Visual C++
  - 1. Memory Management Function (Review)
  - 2. 0xCDs, 0xDDs and 0xFDs in memory
  - 3. _CrtMemBlockHeader
  - 4. _crtDbgFlag

4. _crtDbgFlag(1/15)

- Retrieves and/or modifies the state of the _crtDbgFlag flag to control the allocation behavior of the debug heap manager (debug version only).

- int **_CrtSetDbgFlag**(int newFlag);
  - newFlag: New state for the _crtDbgFlag

4. _crtDbgFlag(2/15)

- **_crtDbgFlag’s** bit fields(位域)

位域	default	说明
_CRTDBG_ALLOC_MEM_DF	On	打开调试分配。当该位为 off 时，分配仍链接在一起，但它们的块类型为 **_IGNORE_BLOCK** 。
_CRTDBG_DELAY_FREE_MEM_DF	Off	防止实际释放内存，与模拟内存不足情况相同。当该位为 on 时，已释放块保留在调试堆的链接列表中，但标记为 **_FREE_BLOCK** ，并用特殊字节值填充**0XDD**。
_CRTDBG_CHECK_ALWAYS_DF	Off	导致每次分配和释放时均调用 **_CrtCheckMemory** 。这将减慢执行，但可快速捕捉错误。
_CRTDBG_CHECK_CRT_DF	Off	导致将标记为 **_CRT_BLOCK** 类型的块包括在泄漏检测和状态差异操作中。当该位为 off 时，在这些操作期间将忽略由运行时库内部使用的内存。
_CRTDBG_LEAK_CHECK_DF	Off	导致在程序退出时通过调用 _CrtDumpMemoryLeaks 来执行泄漏检查。如果应用程序未能释放其所分配的所有内存，将生成错误报告。

4. _crtDbgFlag(3/15)

- Set one or more _crtDbgFlag bit field, and create new state for the flag
  - 1. Call _CrtSetDbgFlag with newFlag equal to _CRTDBG_REPORT_FLAG to obtain the current _crtDbgFlag state and store the returned value in a temporary variable.

```
// Get the current state of the flag
// and store it in a temporary variable
int tmpFlag =
    _CrtSetDbgFlag( _CRTDBG_REPORT_FLAG );
```

4. _crtDbgFlag(4/15)

- Set one or more _crtDbgFlag bit field, and create new state for the flag
  - 2. Turn on any bits by OR-ing the temporary variable with the corresponding bitmasks

```
// Turn On (OR) - Keep freed memory blocks in the
// heap’s linked list and mark them as freed

tmpFlag |= _CRTDBG_DELAY_FREE_MEM_DF;
```

4. _crtDbgFlag(5/15)

- Set one or more _crtDbgFlag bit field, and create new state for the flag
  - 3. Turn off the other bits by AND-ing the variable with a bitwise NOT of the appropriate bitmasks.

```
// Turn Off (AND) - prevent _CrtCheckMemory from
// being called at every allocation request
tmpFlag &= ~_CRTDBG_CHECK_ALWAYS_DF;
```

4. _crtDbgFlag(6/15)

- Set one or more _crtDbgFlag bit field, and create new state for the flag
  - 4. Call _CrtSetDbgFlag with newFlag equal to the value stored in the temporary variable to set the new state for _crtDbgFlag.

```
// Set the new state for the flag
_CrtSetDbgFlag( tmpFlag );
```

4. _crtDbgFlag(7/15)

```
#define _CRTDBG_MAP_ALLOC//which file and line cause leak

#include <stdlib.h>
#include <malloc.h>
#include <crtdbg.h>

int main(int argc, char* argv[]) {
    int tmpFlag = _CrtSetDbgFlag( _CRTDBG_REPORT_FLAG );
    tmpFlag |= _CRTDBG_DELAY_FREE_MEM_DF;
    _CrtSetDbgFlag( tmpFlag );

    char* buffer = (char*)malloc(12);
   free(buffer);
   buffer = NULL;
   return 0;
```

4. _crtDbgFlag(8/15)

- You can avoid giving back the block to the Win32 heap by using the **CRTDBG_DELAY_FREE_MEM_DF** flag to _CrtSetDbgFlag().
- It prevents memory from actually being freed.
- When **CRTDBG_DELAY_FREE_MEM_DF** flag is on, freed blocks are kept in the debug heap's linked list but are marked as _FREE_BLOCK.
  - 通常，所释放的块将从列表中移除。为了检查是否仍在向已释放的内存写入数据，或为了模拟内存不足情况，可以选择在链接列表上保留已释放块。
  - This is useful if you want to detect dangling pointers errors, which can be done by verifying if the freed block is written with 0xDD pattern or something else.

4. _crtDbgFlag(9/15)

DBGHEAP.C __cdecl _free_dbg()

```
/* optionally reclaim memory */
if (!(_crtDbgFlag & _CRTDBG_DELAY_FREE_MEM_DF))  {
  /* remove from the linked list, not delay */
  if (pHead -> pBlockHeaderNext) {
      pHead -> pBlockHeaderNext -> pBlockHeaderPrev =
          pHead -> pBlockHeaderPrev;
  }
  else  {
      _ASSERTE(_pLastBlock == pHead);
      _pLastBlock = pHead -> pBlockHeaderPrev;
  }
```

4. _crtDbgFlag(10/15)

```
if (pHead->pBlockHeaderPrev)  {
    pHead -> pBlockHeaderPrev -> pBlockHeaderNext =
             pHead->pBlockHeaderNext;
 }
else {
     _ASSERTE(_pFirstBlock == pHead);
      _pFirstBlock = pHead->pBlockHeaderNext;
}
```

4. _crtDbgFlag(11/15)

```
    /* fill the entire block including header with dead-land-fill
*/
    memset ( pHead, _bDeadLandFill,
                sizeof( _CrtMemBlockHeader) +
                    pHead->nDataSize + nNoMansLandSize );
   _free_base(pHead);
 }
```

4. _crtDbgFlag(12/15)

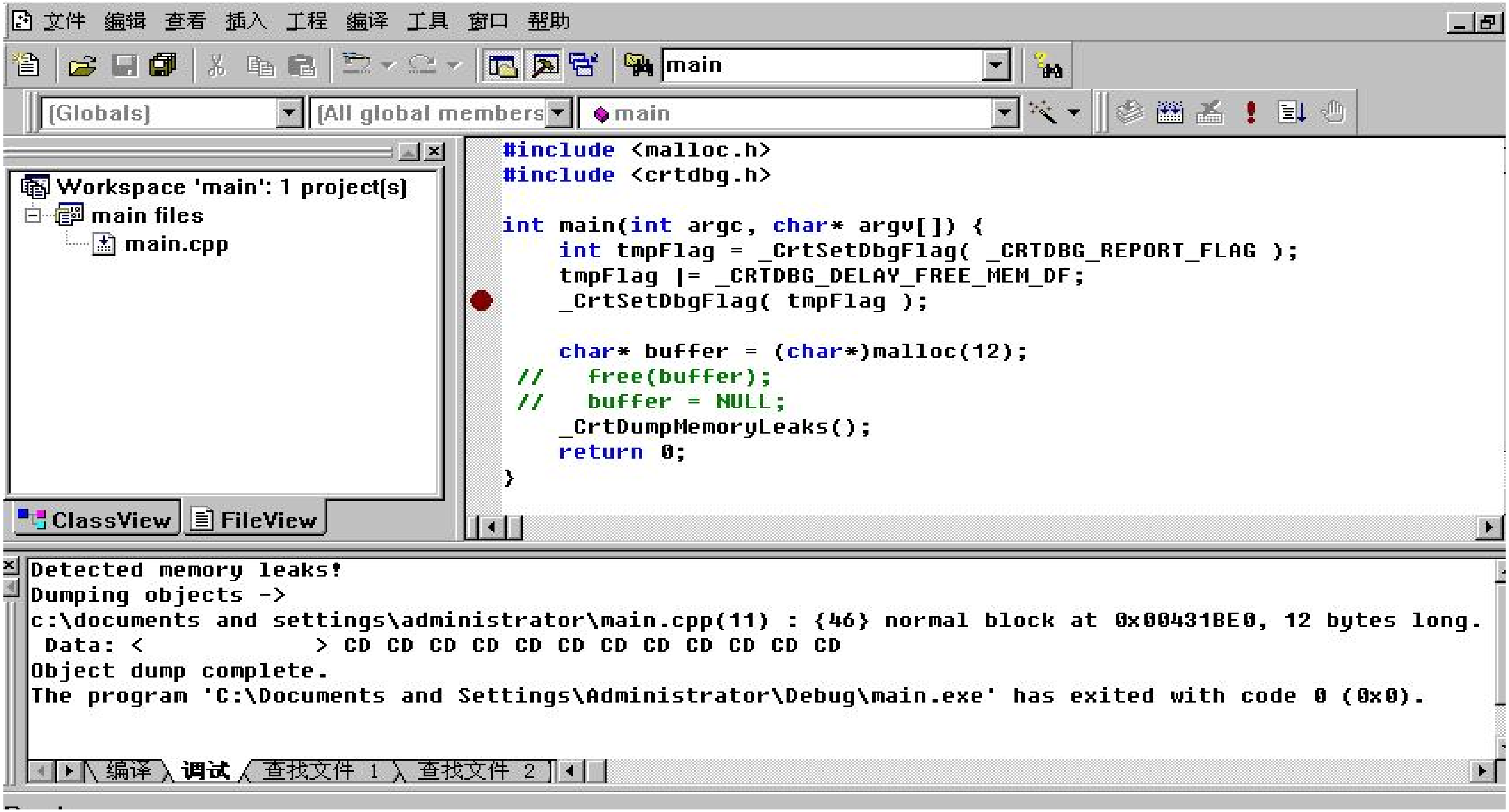
```
else {
     pHead -> nBlockUse = _FREE_BLOCK;

    /* keep memory around as dead space */
   memset ( pbData(pHead),
                   _bDeadLandFill,
                   pHead -> nDataSize);
   }
}
```

4. _crtDbgFlag(13/15)

```
#define _CRTDBG_MAP_ALLOC //which file and line cause leak
#include <stdio.h>
#include <malloc.h>
#include <crtdbg.h>
int main(int argc, char* argv[]) {
// method1 set flag
//int tmpFlag = _CrtSetDbgFlag( _CRTDBG_REPORT_FLAG );
//   tmpFlag |= _CRTDBG_LEAK_CHECK_DF;
//   _CrtSetDbgFlag( tmpFlag );

     char* buffer = (char*)malloc(12);

     //method 2
     _CrtDumpMemoryLeaks();
   return 0;
}
```

4. _crtDbgFlag(14/15)

- The program will output the following message on VC debug mode:

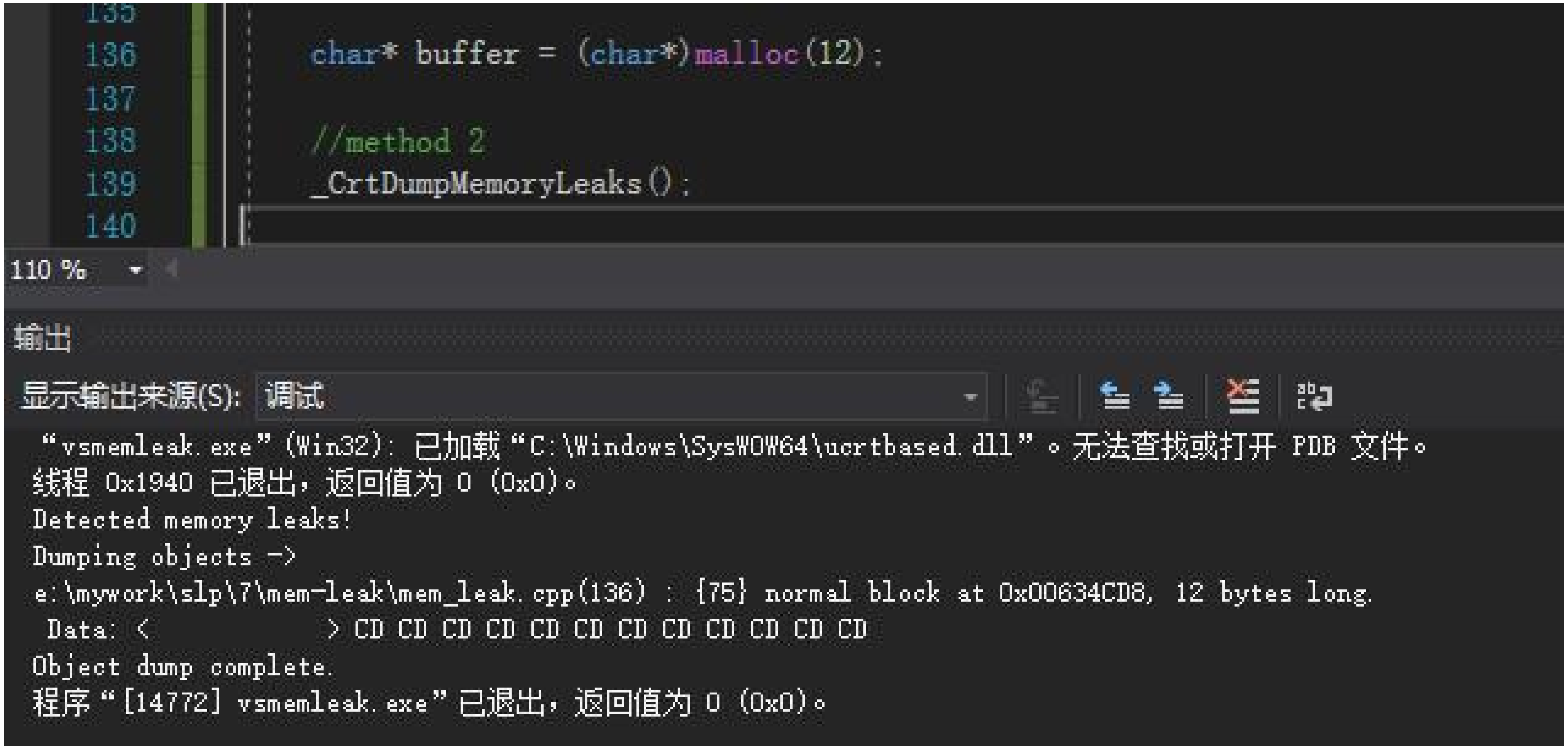


```
#include <malloc.h>
#include <crtdbg.h>

int main(int argc, char* argv[]) {
    int tmpFlag = _CrtSetDbgFlag( _CRTDBG_REPORT_FLAG );
    tmpFlag |= _CRTDBG_DELAY_FREE_MEM_DF;
    _CrtSetDbgFlag( tmpFlag );

    char* buffer = (char*)malloc(12);
 //    free(buffer);
 //    buffer = NULL;
    _CrtDumpMemoryLeaks();
    return 0;
}
```

```
Detected memory leaks!
Dumping objects ->
c:\documents and settings\administrator\main.cpp(11) : {46} normal block at 0x00431BE0, 12 bytes long.
 Data: <            > CD CD CD CD CD CD CD CD CD CD CD CD
Object dump complete.
The program 'C:\Documents and Settings\Administrator\Debug\main.exe' has exited with code 0 (0x0).
```

4. _crtDbgFlag(14/15)

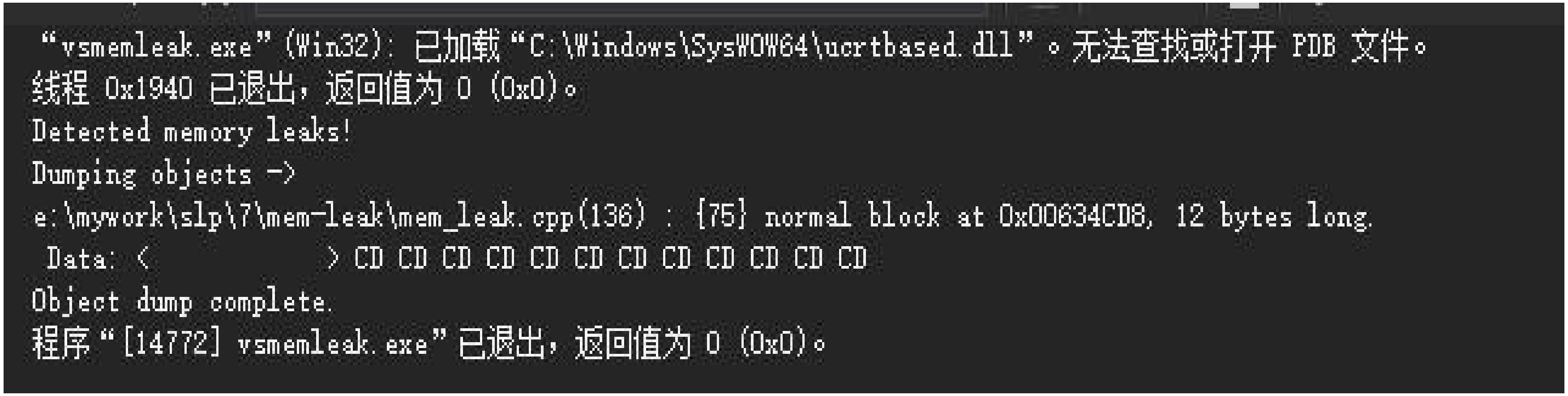
- The program will output the following message on VS debug mode:
- #define _CRTDBG_MAP_ALLOC// 否则没有文件名行号



```
    char* buffer = (char*)malloc(12);

    //method 2
    _CrtDumpMemoryLeaks();
```

```
 “vsmemleak.exe”(Win32): 已加载“C:\Windows\SysWOW64\ucrtbased.dll”。无法查找或打开 PDB 文件。
线程 0x1940 已退出，返回值为 0 (0x0)。
Detected memory leaks!
Dumping objects ->
e:\mywork\slp\7\mem-leak\mem_leak.cpp(136) : {75} normal block at 0x00634CD8, 12 bytes long.
 Data: <            > CD CD CD CD CD CD CD CD CD CD CD CD
Object dump complete.
程序“[14772] vsmemleak.exe”已退出，返回值为 0 (0x0)。
```

4. _crtDbgFlag(15/15)



```
“vsmemleak.exe”(Win32): 已加载“C:\Windows\SysWOW64\ucrtbased.dll”。无法查找或打开 PDB 文件。
线程 0x1940 已退出，返回值为 0 (0x0)。
Detected memory leaks!
Dumping objects ->
e:\mywork\slp\7\mem-leak\mem_leak.cpp(136) : {75} normal block at 0x00634CD8, 12 bytes long.
 Data: <            > CD CD CD CD CD CD CD CD CD CD CD CD
Object dump complete.
程序“[14772] vsmemleak.exe”已退出，返回值为 0 (0x0)。
```

e:\myworks\slp\7\mem-leak.cpp(136)显示分配泄漏内存的文件名，以及文件名后括号中的数字表示发生泄漏的代码行号

{xx}：花括弧内的数字是内存分配序号，本文例子中是 {75}；

normal block：内存块的类型；

用十六进制格式表示的内存位置，如：at 0x00634cd8；

以字节为单位表示的内存块的大小，如：12 bytes long；

前 16 字节的内容（也是用十六进制格式表示），如：Data: <CD>

Summary

- Implementation Issues
  - How much memory for allocation?
  - How do we know how much memory to free just given a pointer?
  - How do we keep track of the free blocks?