Practical Foundations of Debugging

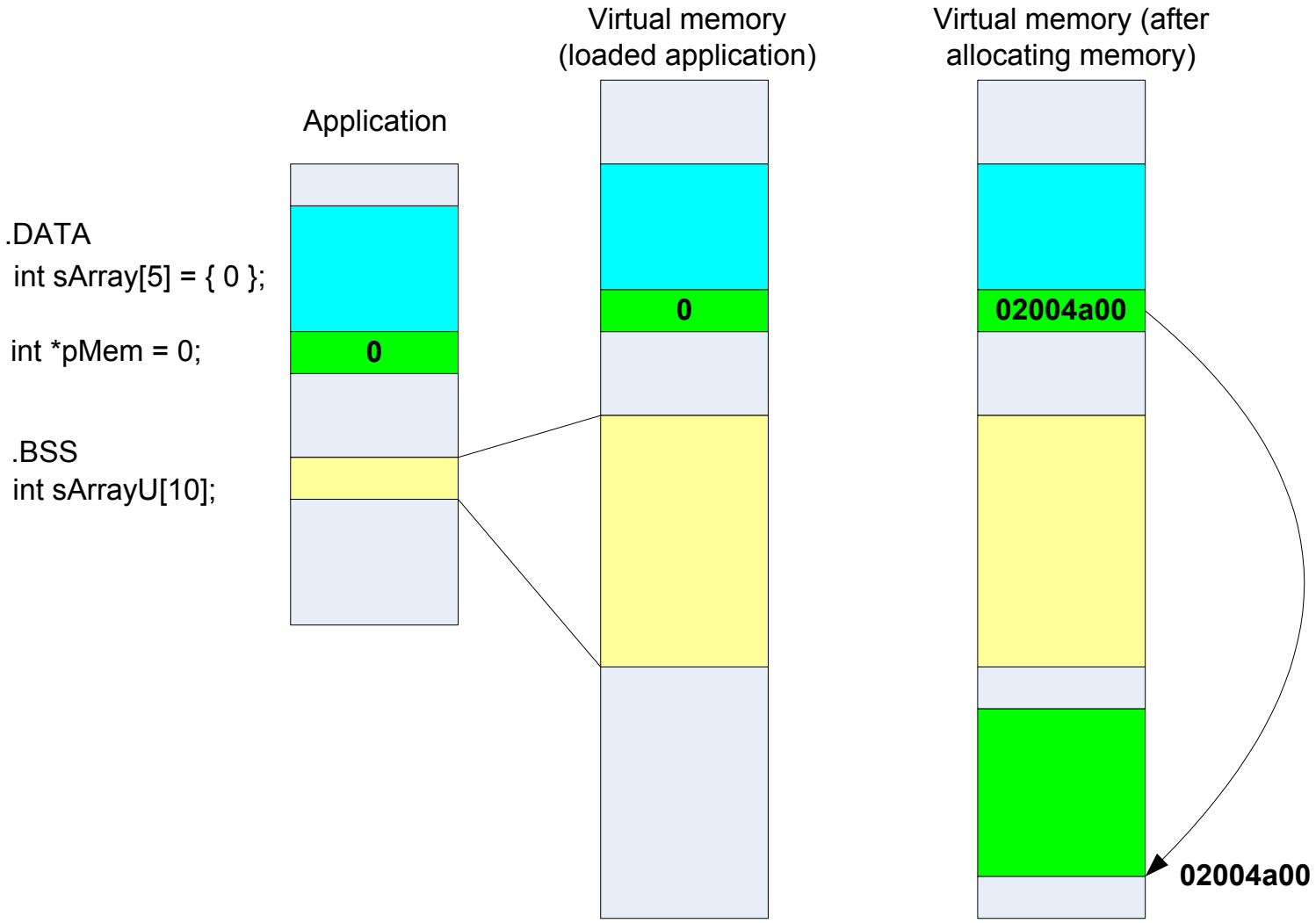
Chapter 10

Arrays and Structures in Memory – Part 2

Dynamic Arrays and Heap

Designed, written and illustrated by Dmitry Vostokov

Static vs. Dynamic Memory



Memory Managers

- General-purpose (variable-size)
  - Performance
  - Fragmentation
- Fixed-size
  - Flexibility
- Mixed (combination of previous two)
- Garbage collection
  - Compatibility

Layered Memory Management

- Kernel
- OS (high-level abstractions)
- Compiler's default run-time
- C++ standard library containers
- User-defined containers

Fragmentation

General-purpose allocation

- Housekeeping information
- Padding, alignment
- How ‘free’ frees a pointer?

Checking for memory allocation failures

- Useless!
- DoS attack on buffer lenghts

Rules for reading declarations

1. Brackets take precedence
2. If you see ‘[]’ read to the write and add ‘array of’
3. Read to the left and if you see ‘*’ add ‘pointer to’ else add ‘of’ and put type name. Goto #1

```
int *p;      // pointer to int
int a[N];    // array of ints
int *ap[N]; // array of pointers to int
int **pp;    // pointer to pointer to int
int **app[N]; // array of pointers to pointer to int
int (*pa)[N]; // pointer to array of ints
int *(*apap[N])[N]; // array of pointers to array of
    pointers to int
```

Pointers to array

```
int (*pa)[10]; // pointer to array with 10 integers
int array10[10];
int array11[11];

pa = &array10;
pa = &array11;    // ERROR, two different objects

printf("%p - %p\n", pa, array10);
```

Conclusion: array10 and &array10 have the same value (r-value)

Why do we need dynamic arrays?

Example: reading information from file
File has the following layout:
- DWORD: number of elements
- DWORD[]: an array of integers

```
File: 2, 0, 1
File: 10, 0, 1, 0, 1 , 0, 2, 2, 2, 2, 2

const int MAXDATA = 1000;
int data[MAX_DATA];
int numElements;

void ReadData()
{
    ReadFile(…, &numElements, sizeof(numElements), …);
    if (numElements > sizeof(data)/sizeof(data[0]))
    {
          numElements = sizeof(data)/sizeof(data[0]);
    }
    ReadFile(…, data, numElements*sizeof(data[0]), …); // reading bytes
}
```

So we have a dilemma:
Small file – waste of memory
Big file (more than 1000 elements) – not enough space, data is truncated

Solution: dynamic arrays

File has the following layout:

- DWORD: number of elements
- DWORD[]: an array of integers

```
File: 2, 0, 1
File: 10, 0, 1, 0, 1 , 0, 2, 2, 2, 2, 2
File: 0

// C/C++
int *data;
int numElements;

void ReadData()
{
    free(data);      // NULL can be passed

    ReadFile(…, &numElements, sizeof(numElements), …);
    data = (int *)malloc(numElements); // you can pass 0 here, needs a cast in C++

    if (data)
    {
          ReadFile(…, data, numElements*sizeof(data[0]), …);
    }
}
```

dynamic arrays in C++

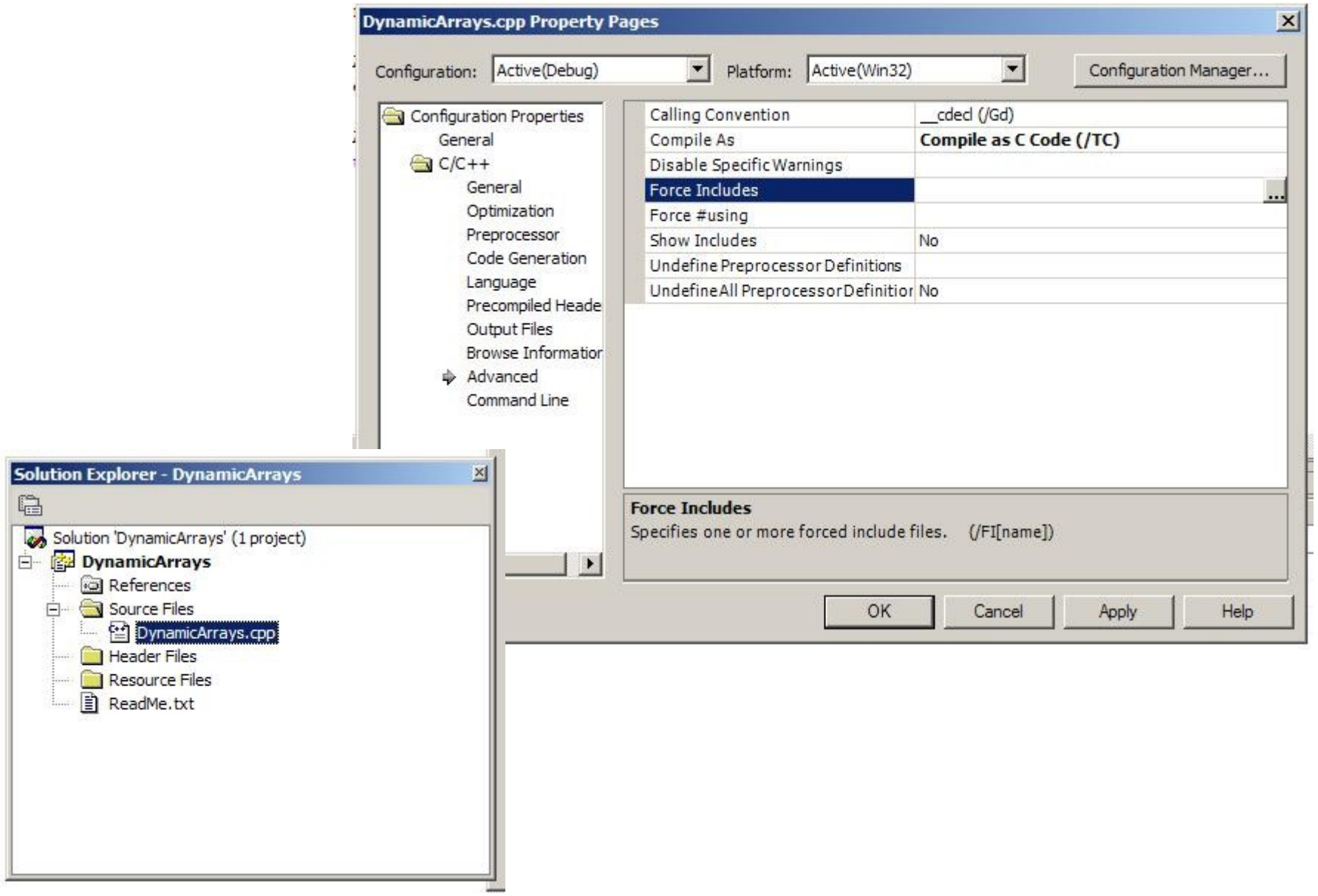
```
// C++
int *data;
int numElements;

void ReadData()
{
   delete [] data;   // [] are important here, NULL can be passed

   ReadFile(…, &numElements, sizeof(numElements), …);
   data = new int[numElements]; // you can pass 0 here

   if (data)
   {
        ReadFile(…, data, numElements*sizeof(data[0]), …);
   }
}
```

Compiling the same file as C or C++ code in VC++.NET



Example

```
#include <stdlib.h>

int *dynArray;
int *pInteger;

int main(int argc, char* argv[])
{
    // allocating an integer in malloc style
    pInteger = (int *)malloc(1);
    int integer = *pInteger;
    free(pInteger);

    // allocating an array in malloc style
    dynArray = (int *)malloc(100);
    integer = dynArray[10];
    free(dynArray);

    // allocating an integer in C++ style
    pInteger = new int[1];
    integer = *pInteger;
    delete pInteger;

    // allocating an array in C++ style
    dynArray = new int[100];
    integer = dynArray[10];
    delete [] dynArray;

    return 0;
}
```

What's next?

- Arrays and structures – parts 3 and 4
- Virtual memory and paging
- Multithreading, memory and stacks
- Calling Windows functions
  (stdcall vs. cdecl)
- Classes and Objects
- Templates
- Strings
- Pointers to pointers (LPSTR *)