Diagnosing Memory Leaks

memory leak — the sin of losing access to dynamically-allocated memory before it has been deallocated

Basic scenarios:

- Allocated and deallocated within same scope
- Allocated in one scope and deallocated in another scope
- Auto-allocated via a class constructor or other member function
- Auto-deallocated via a class destructor

None of these scenarios should be that difficult to handle reliably, especially if dynamic allocation is restricted to objects whose destructors then have the responsibility of cleaning it up…

… unfortunately, experience indicates otherwise.

©2003 McQuain WD

Trivial Example

```
#include <iostream>
#include <new>
using namespace std;
#include <stdlib.h>

void F();

int main() {

   int* p = new int[1000];   // allocate 4 * sizeof(int) bytes
   F();

   return 0;
}

void F() {

   int* p = new int[100];    // allocate 4 * sizeof(int) bytes
}
```

There are two obvious leaks in the given program, totaling 4400 bytes. That's not a big leak, but that is also irrelevant.

©2003 McQuain WD

Automated Detection of Leaks

Visual C++ .NET provides a fairly simple and fairly accurate way to diagnose memory leaks.

The first step is to define a symbol to turn on a special debug version of the heap functions that manage memory allocation:

```
#define CRTDBG_MAP_ALLOC
```

The second step is to include a non-Standard header that declares some Windows-specific diagnostic functions:

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

The third step is to call a diagnostic function to report any detected leaks:

```
_CrtDumpMemoryLeaks();
```


©2003 McQuain WD

Revised Code

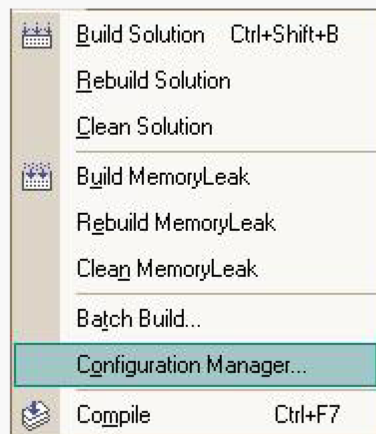
```
#include <iostream>
#include <new>
using namespace std;

#define CRTDBG_MAP_ALLOC  // selects debug version of heap functions
#include <stdlib.h>
#include <crtdbg.h>       // non-Standard header for memory diagnostics

void F();

int main() {

   int* p = new int[1000];        // allocate 4 * sizeof(int) bytes
   F();

   return _CrtDumpMemoryLeaks();  // call diagnostic function
}

void F() {

   int* p = new int[100];         // allocate 4 * sizeof(int) bytes
}
```

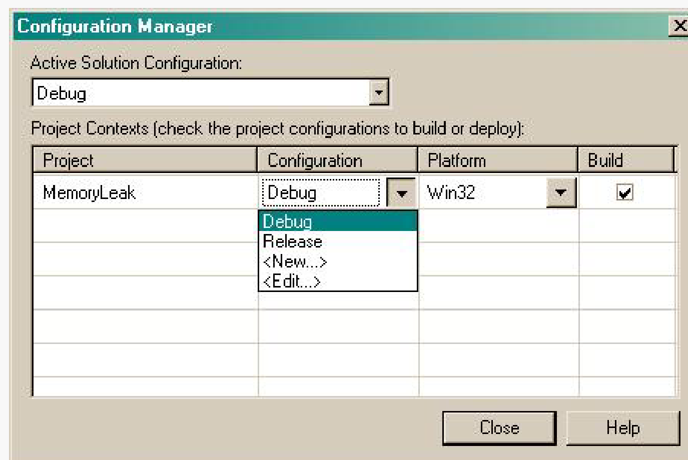
The call to the diagnostic function is embedded in the `return` statement so that we can be sure that all relevant destructors have fired before it is called.

©2003 McQuain WD

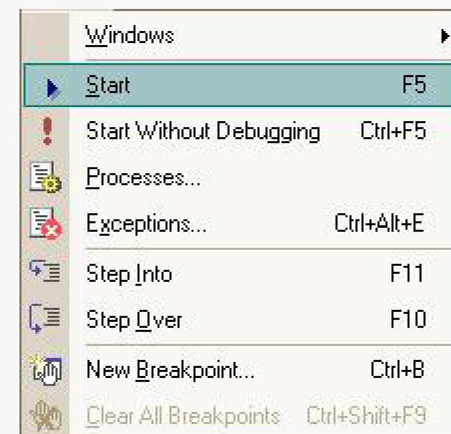
Running the Diagnostics

In order for the diagnostics to produce results, you must build the executable in debug mode:





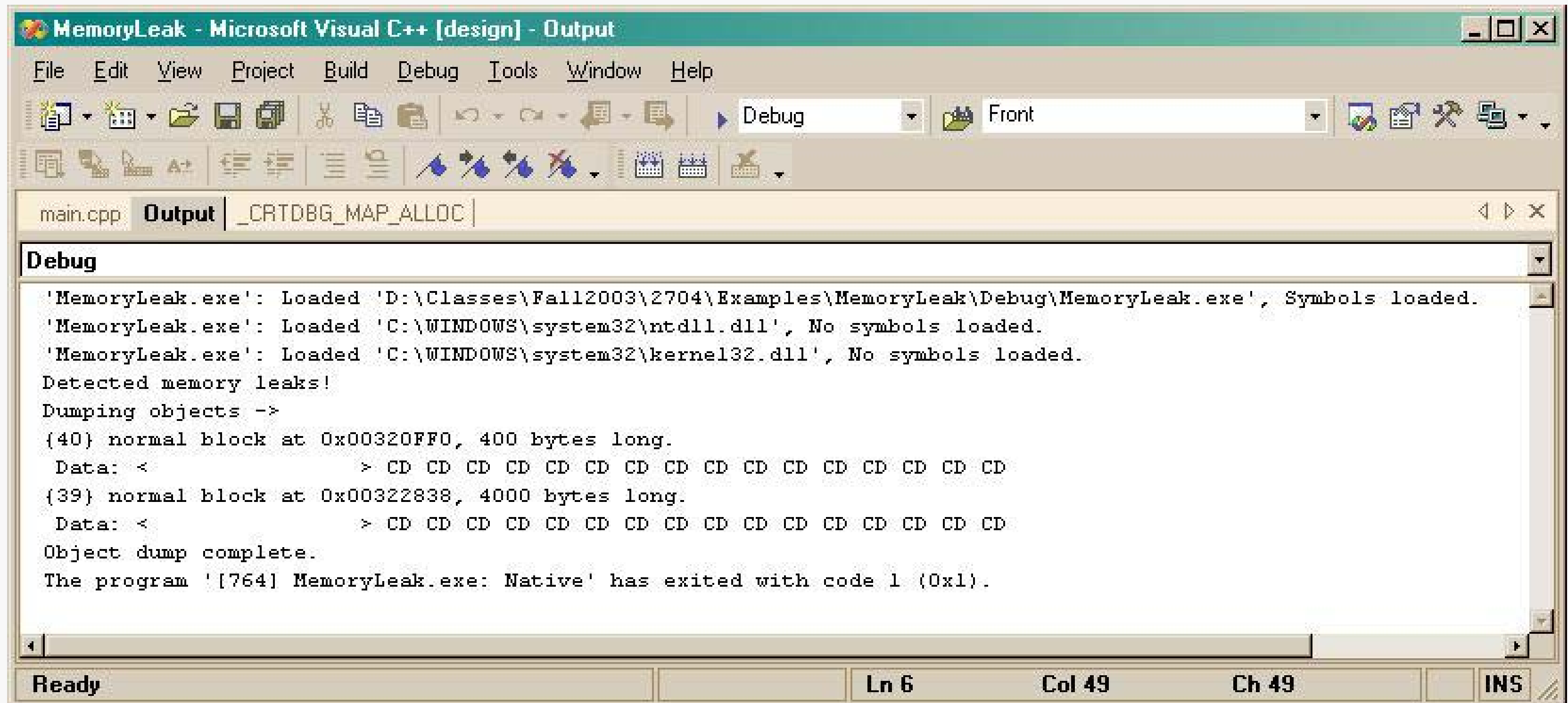
And… you must execute the program with debugging information on:



©2003 McQuain WD

Report

The diagnostic report is shown by default in the IDE Output window:



Note that the report does diagnose both leaks, in this case.

©2003 McQuain WD