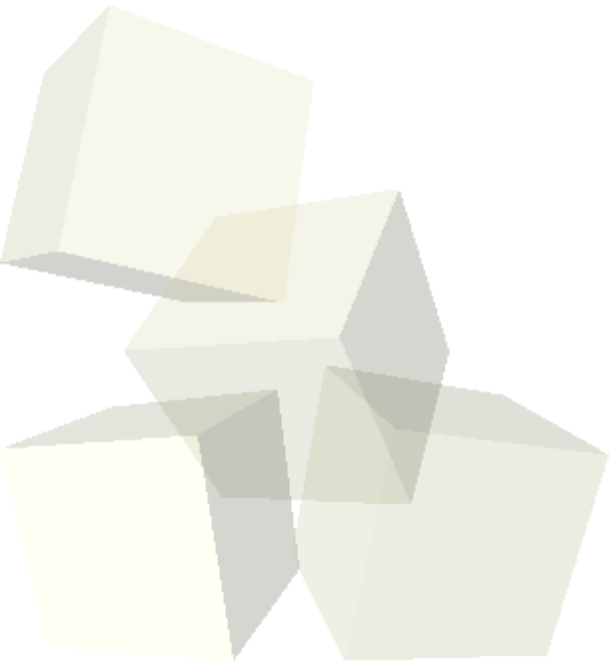




The C++ Memory is Smart.
Are you too?





- **Memory leaks** → out of memory exceptions → program fail (may cause crash in other running programs)
- **Wrong pointers** → exceptions, corruption → wrong behavior or program crash
- **Buffer overflow** → memory corruption → undefined behavior





- **Memory leaks** → out of memory exceptions → program fail (may cause crash in other running programs)
- **Wrong pointers** → exceptions, corruption → wrong behavior or program crash
- **Buffer overflow** → memory corruption → undefined behavior

is it any good?





Prevent future troubles

- Use **smart pointers** as possible:

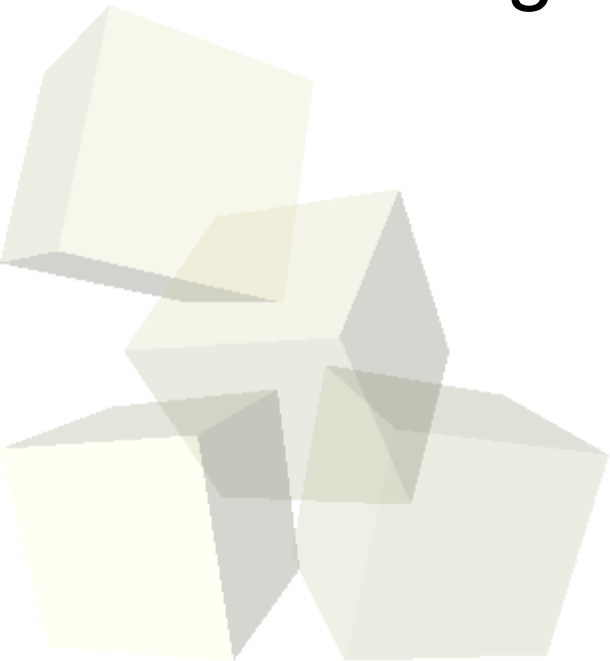
```
boost::shared_ptr<CObject> CObjectPtr;  
CComPtr<IHTMLDocument2> spHtmlDoc;
```

- Assign NULL to all **deleted** pointers right after **delete** (and to all its copies if exist)
- Use Secure CRT instead of old-style usage:
`sprintf` → `sprintf_s`
- Use debugger, **AppVerifier**, etc. to detect bad memory usage



Already in trouble

Fighting with the **Wrong** pointers!





Solution #1

- `bool ClassOne::getSomeBool()`
`{`
 `return true;`
`}`

```
bool SomeClass::anotherMethod()
{
    try // m_pOne is ClassOne*
    {
        return m_pOne->getSomeBool();
    }
    catch(...)
    {
        return true; // or false
    }
}
```

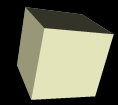


Never ever do that!

- `bool ClassOne::getSomeBool()`
`{`
 `return true;`
`}`

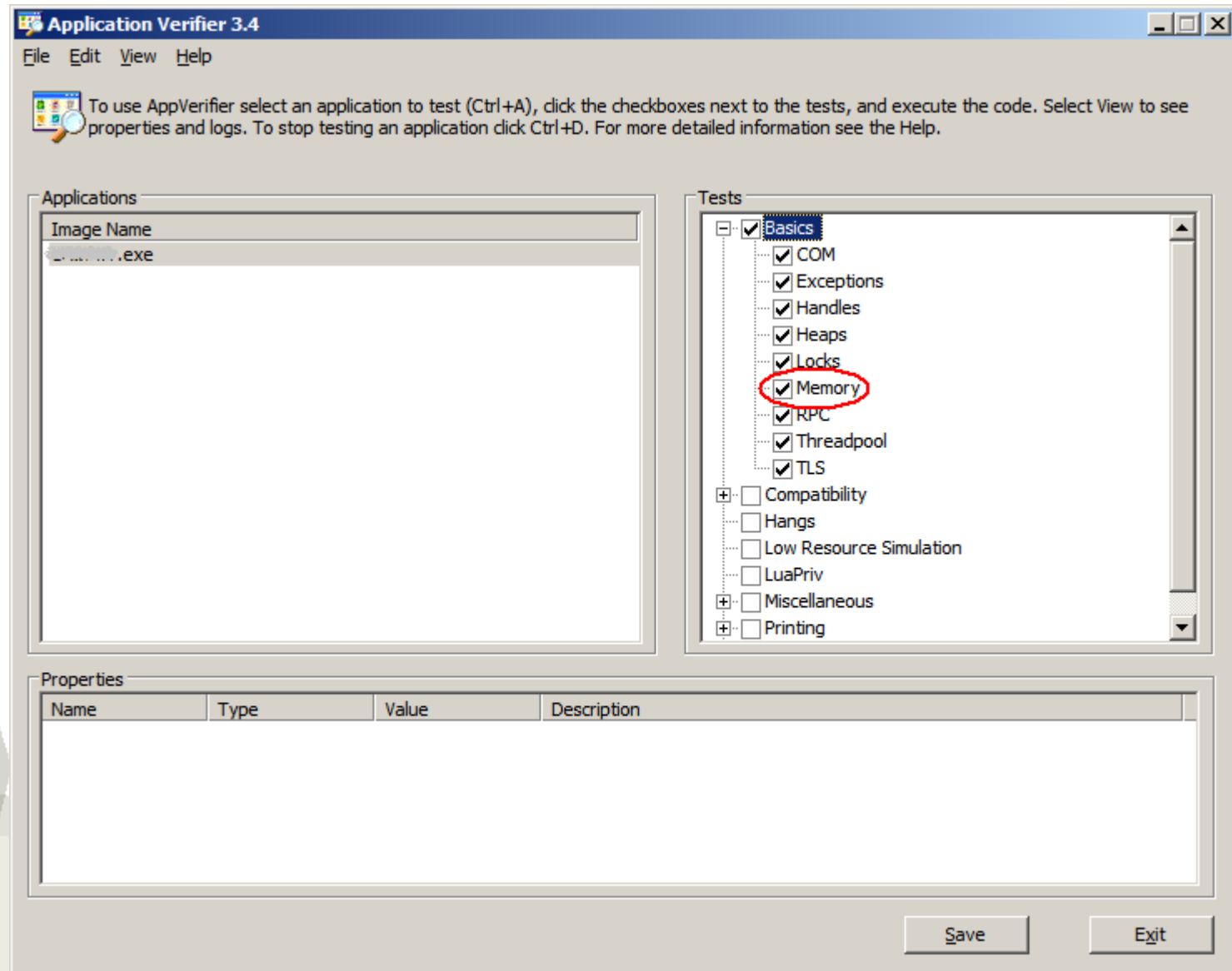
```
bool SomeClass::anotherMethod()
{
    try // m_pOne is ClassOne*
    {
        return m_pOne->getSomeBool();
    }
    catch(...)
    {
        return true; // or false
    }
}
```

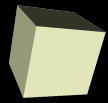




You did it (and not the single time)

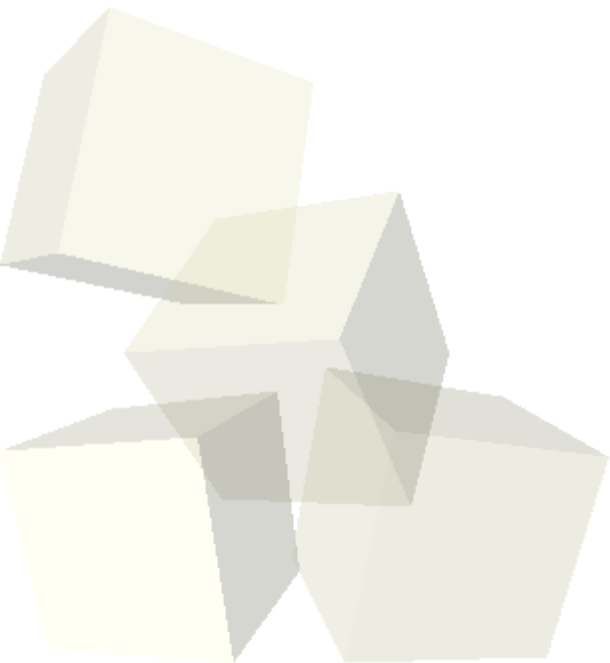
- Use AppVerifier, memory check section





Is that information useful for you?

- First-chance exception at 0x0058e063 in TestApp.exe:
0xC0000005: Access violation reading location 0xcdcdcdcd.





Yes, of course

- First-chance exception at **0x0058e063** in TestApp.exe:
0xC0000005: Access violation reading location 0xcdcdcdcd.

=====

VERIFIER STOP 00000002 : pid 0xC58: Access violation exception.

CDCDCDCD : Invalid address causing the exception

0058E063 : Code address executing the invalid access

0012F9A8 : Exception record

0012F9C4 : Context record

=====

This verifier stop is not continuable. Process will be terminated



Debug CRT Heap internals

- 0xCDCDCDCD

clean data, just allocated (new/malloc), but not initialized yet

- 0xDDDDDDDD

dead data, after free/delete

- 0xFDFDFD

Fences around memory, “No mans land”

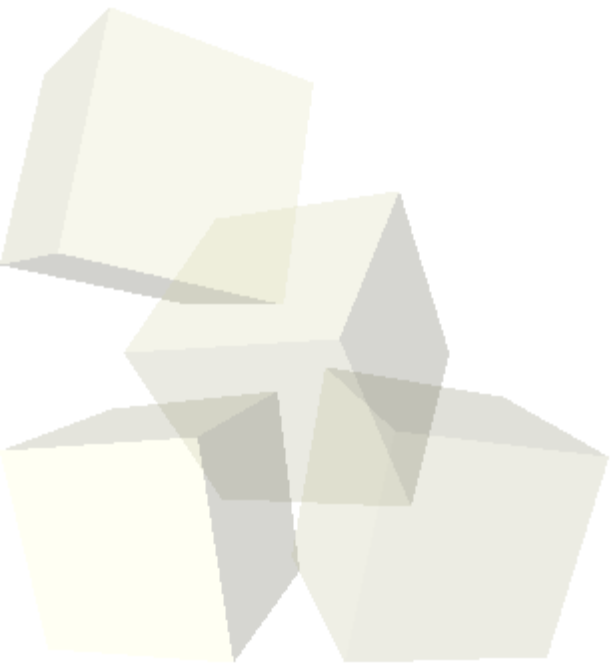
- ♦ Helps finding buffer overflows also

- After HeapAlloc() internal: 0xBAADF00D



Internals helping find out errors

- First-chance exception at 0x0058e063 in TestApp.exe:
0xC0000005: Access violation reading location
0xCDCDCDE3.





Internals helping find out errors

- First-chance exception at 0x0058e063 in TestApp.exe:
0xC0000005: Access violation reading location
0xCDCDCDE3.

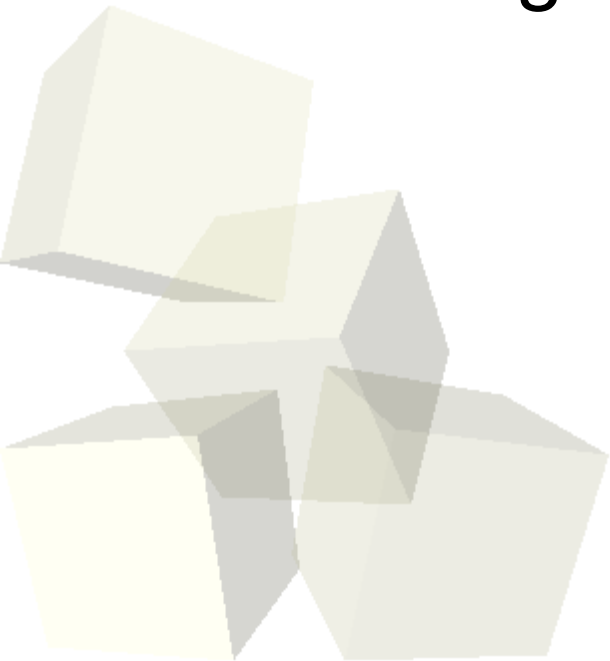
- ```
bool SomeClass::anotherMethod()
{
 try // m_pOne is ClassOne*
 {
 return m_pOne->getSomeBool();
 }
}
```

- The m\_pOne local field is not initialized with some value → it is defaulted to 0xCDCDCDCD
- Trying to access its virtual member with offset of 22, hence got 0xCDCDCDE3



# Already in trouble

Fighting with the **Memory Leaks!**



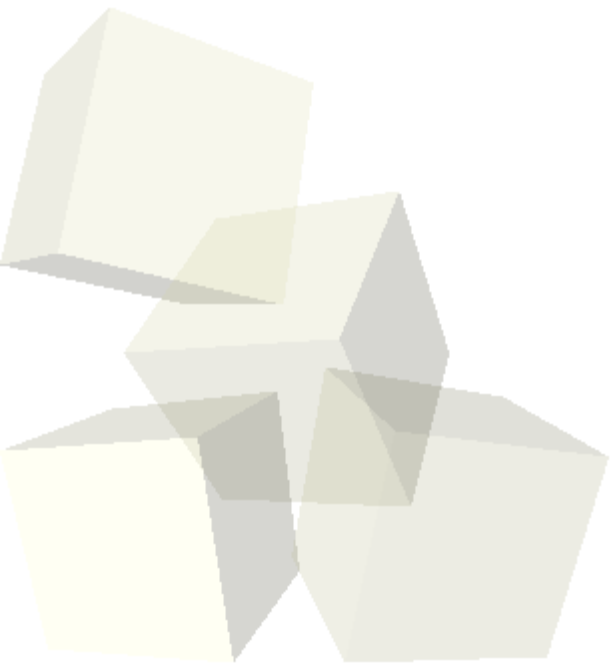


```
■ class CbinTree
```

```
private:
```

```
boost::shared_ptr<CbinTree> CSubTreeLeft;
```

```
boost::shared_ptr<CbinTree> CSubTreeRight;
```





# Smart Pointers, really smart?

■ `class CbinTree`

`private:`

`boost::shared_ptr<CbinTree> CSubTreeLeft;`  
`boost::shared_ptr<CbinTree> CSubTreeRight;`

■ `class CNBaseTree`

`public:`

`CNBaseTree(int base)`

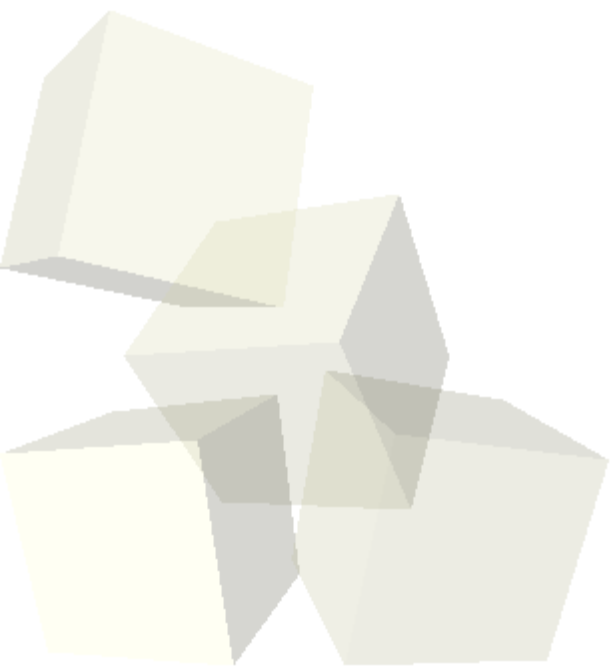
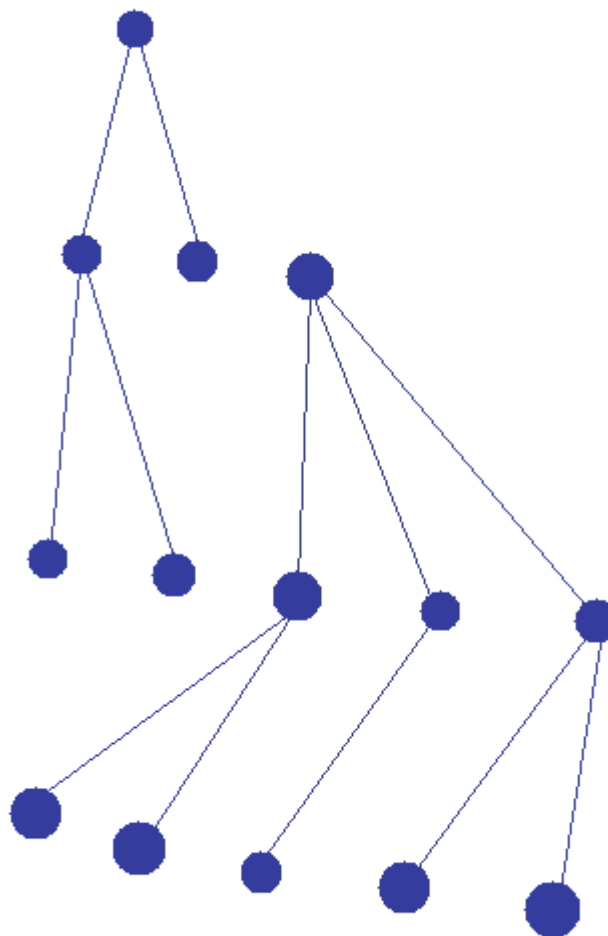
`private:`

`???`



# What a bad tree!

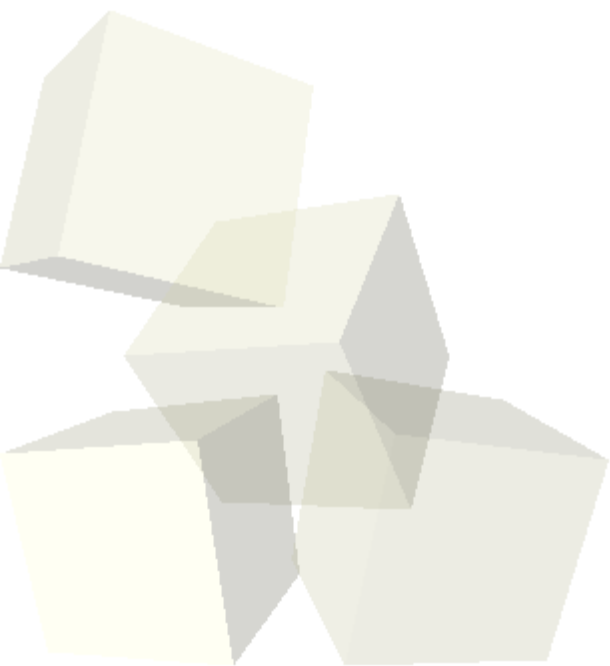
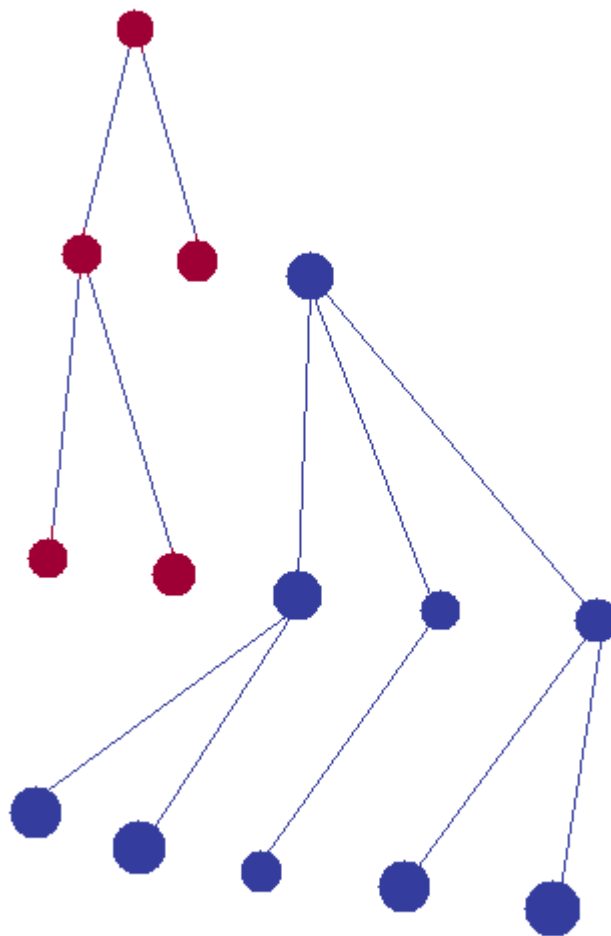
- After some operations:





# What a bad tree!

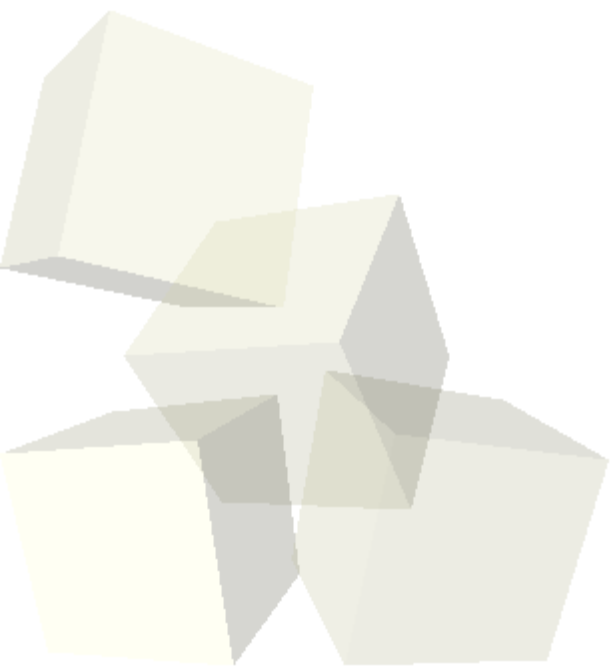
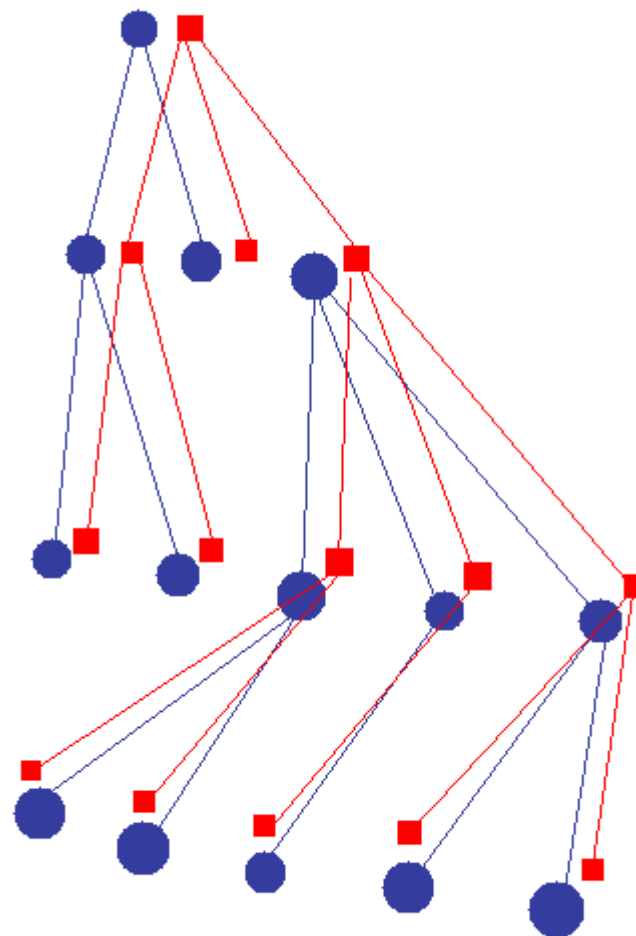
- After delete attempt:

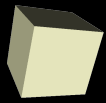




# More memory, more correctness

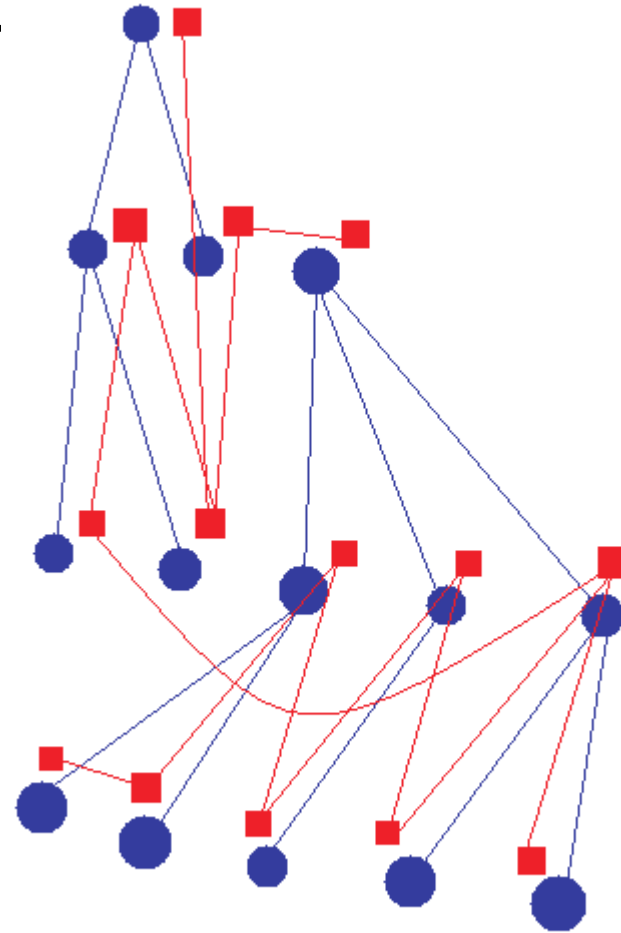
- Right squares are local memory managers:
- Created on sub-element create,  $O(1)$

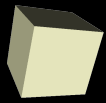




# More memory, more correctness

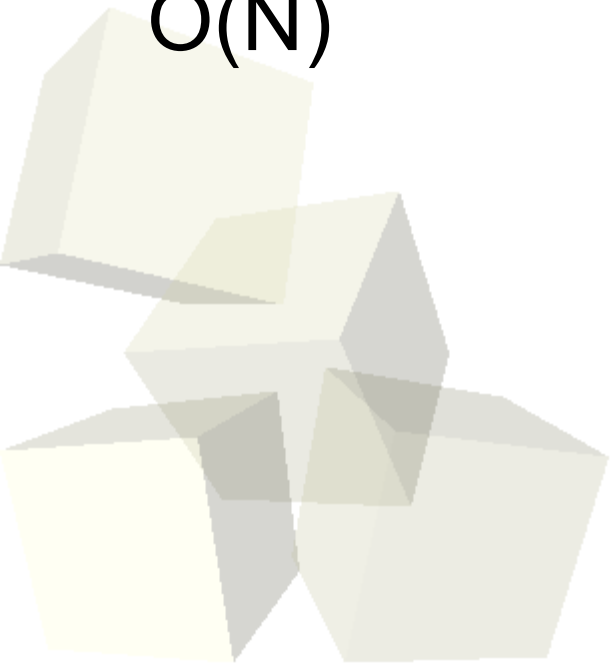
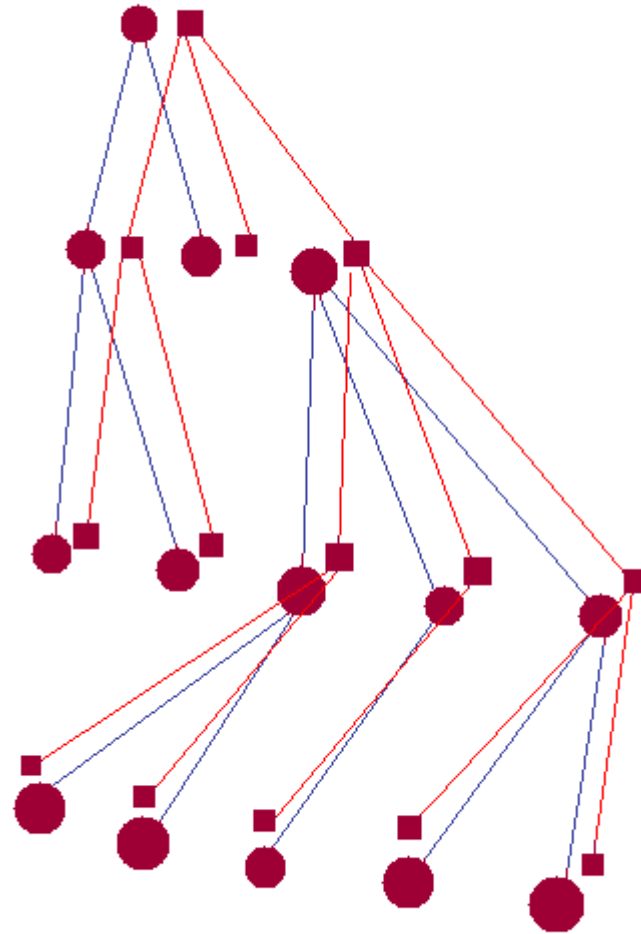
- Right squares are local memory managers:
- Created on sub-element create,  $O(1)$
- Creation order may be not the same as current structure representation





# More memory, more correctness

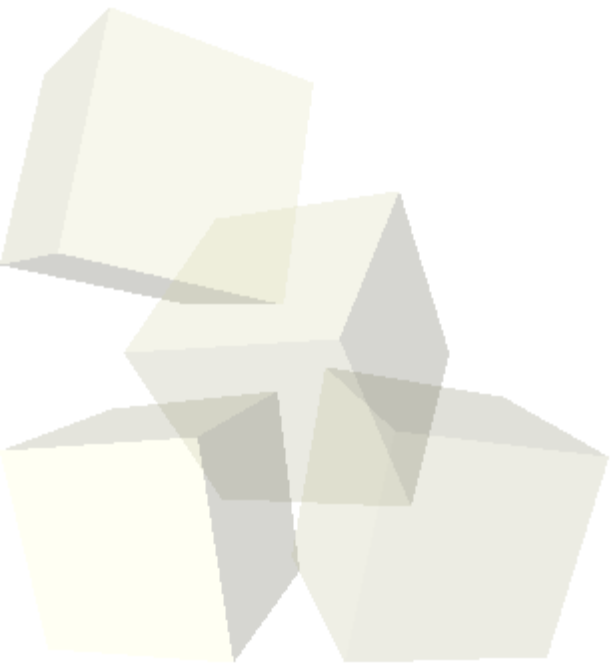
- Right squares are local memory managers:
- Created on sub-element create,  $O(1)$
- On root dispose, delete all children,  $O(N)$





# No silver bullet again

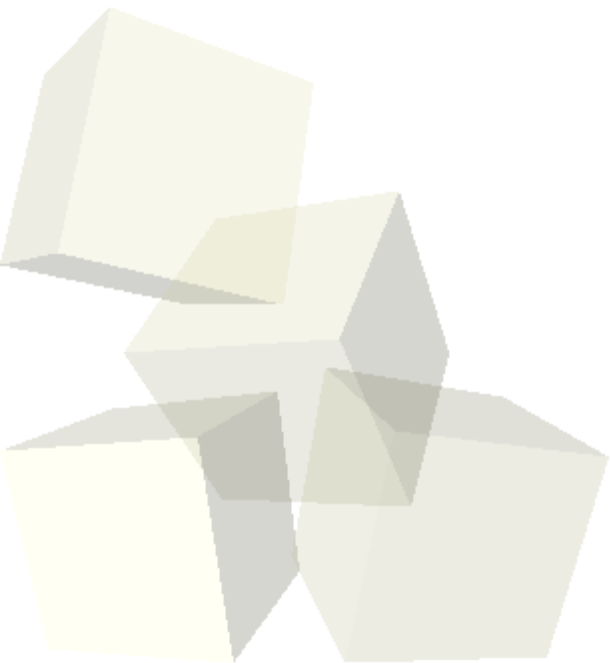
- Good for temporary isolated operations:
  - ♦ Data pre-process, save, load
  - ♦ For programs working with the “projects”
- Bad for long-term structures (we use additional memory and really do nothing)





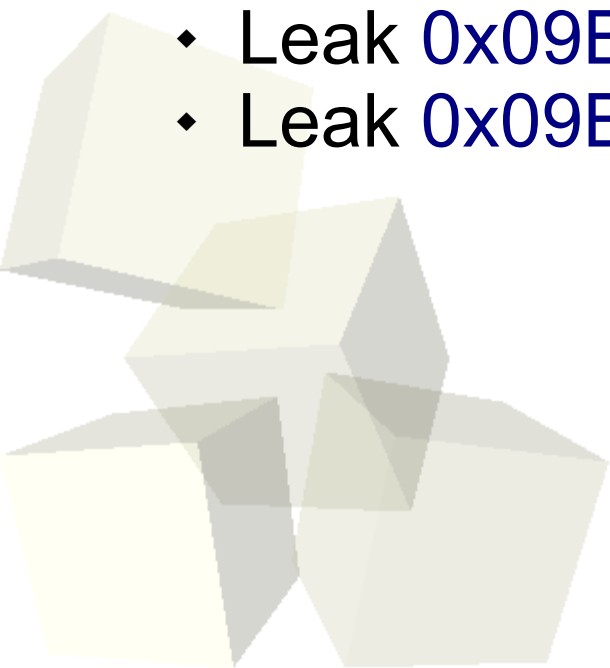
# Already in trouble

Dealing with the problem  
(not with the side-effects)





- Leak 0x09B92F80
  - ♦ Thread 0015f0 <time 277606968>  
CSchemaHandler::ProcessSchemaAttribute :  
XmlInfo::CAttribute() [Schema.cpp, line 577]  
created pointer 0x09B92F80, parent is 0x08C92F98
- Schema.cpp, line 577
  - ♦ Leak 0x09B92F80
  - ♦ Leak 0x09B95A90
  - ♦ Leak 0x09BF1B22





# How to reach, step #1

- `#ifdef DEBUG`  
`#include <stdlib.h>`  
`#include <crtdbg.h>`  
`#define _CRTDBG_MAP_ALLOC`  
`#endif`
- `#ifdef DEBUG`  
`_CrtDumpMemoryLeaks();`  
`#endif`



- Detected memory leaks!  
Dumping objects ->

{8898} normal block at 0x0A497B30, 28 bytes long.

Data: < ^ > E0 7F C7 03 E0 7F C7 03 E0 7F C7  
03 9E 05 5E 00

{8459} normal block at 0x0A4B7128, 176 bytes long.

Data: <C : \ D o c u m > 43 00 3A 00 5C 00 44 00 6F 00  
63 00 75 00 6D 00

{4350} normal block at 0x09EFAFD8, 32 bytes long.

Data: < 0 7 2 0 = 8 5 > 1D 04 30 04 37 04 32 04 30 04  
3D 04 38 04 35 04



# How to reach, step #2

## ■ Define new NEW:

- `#define NEW(x) log_new_pass(new x,#x, __FUNCTION__,this)`

```
template<class T> T log_new_pass(T data, LPCSTR what,
 LPCSTR func, void* parent)
{
 logPointerOperation(func, what, data, parent);
 return data;
}
```

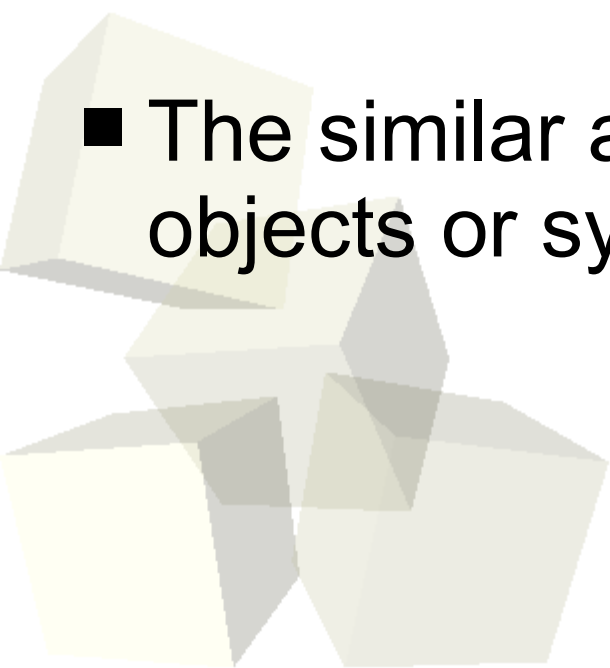
## ■ Use new NEW:

- Replace all “new” → “NEW”
- `#define new NEW`



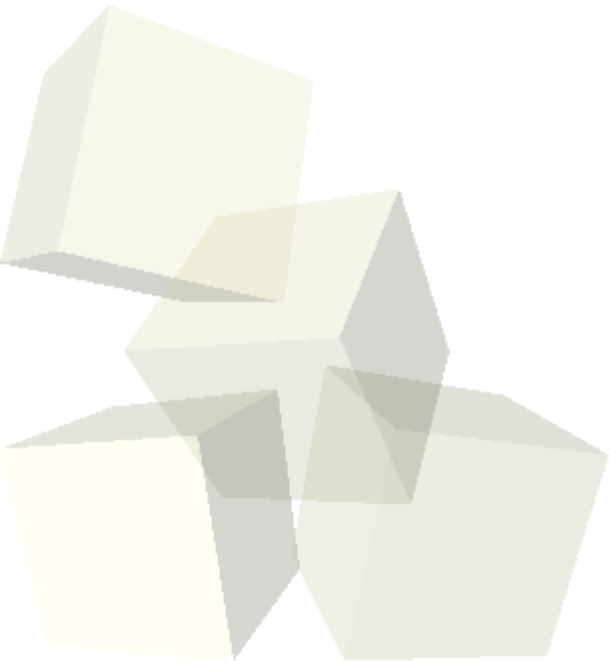
# How to reach, step 3

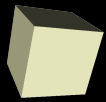
- Match the **new** and **leaks** logs
  - ◆ Of course, automatically:
    - `match_log.exe news.txt leaks.txt > dump.txt`
  
- Just Fix and be happy
  
- The similar approach may be used for COM objects or system handles





Dispose arrays easily





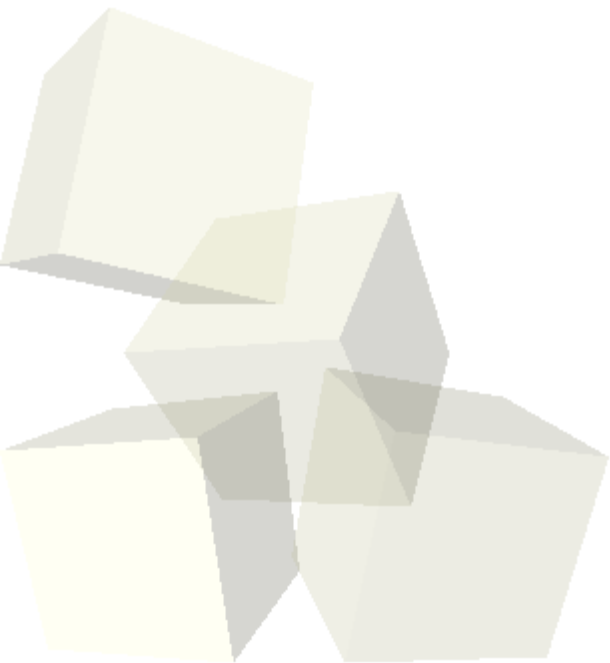
# Working with dynamic arrays...

```
■ char* pBuf = new char[ulFileSize];
 if (!File.Open(name))
 {
 delete []pBuf;
 pBuf = NULL;
 return false;
 }
 if (!File.Read(pBuf))
 {
 delete []pBuf;
 pBuf = NULL;
 File.Close();
 return false;
 }
```



```
■ char* pBuf = new char[ulFileSize];
 ArrayFinalizator fin(&pBuf);

 if (!File.Open(name)) return false;
 if (!File.Read(pBuf)) { File.Close(); return false; }
```





- `char* pBuf = new char[ulFileSize];`  
`ArrayFinalizator fin(&pBuf);`  
  
`if (!File.Open(name)) return false;`  
`if (!File.Read(pBuf)) { File.Close(); return false; }`
- Why `fin(&pBuf)`, not `fin(pBuf)`?
- Designed for not to fail if `pBuf` deleted manually:

```
delete []pBuf;
pBuf = NULL;
return true;
}
```



# Not so complex

```
■ template<class T> class ArrayFinalizator {
public:
 ArrayFinalizator(T** object)
 {
 m_object = object;
 }
 virtual ~ArrayFinalizator()
 {
 if (m_object && *m_object)
 {
 delete[] *m_object;
 *m_object = NULL;
 }
 }
private:
 T** m_object;
};
```



Thanks for attention!

