CS 485
Advanced Object Oriented Design

Dynamic Memory

Spring 2019

pointers! new/delete

- Dynamic memory

Don’t use malloc/free in C++

- Design Techniques
  - `RAII`
    - resource acquisition is initialization
      - constructor/destructor
  - CopyAndSwap

const

```
const int * pData;

int const * pData;

int * const pData;

const int * const pData;

int const * const *pData;
```

const applies to the type to the immediate left, however you can put const as the left most token and then it applies to the immediate right.

http://c-faq.com/decl/spiral.anderson.html

NULL vs nullptr

- NULL is really an int
- nullptr is its own type (std::nullptr_t)

```
void foo(int *ptr);
void foo(int data);

foo(NULL); // who gets called?
foo(nullptr); // who gets called?
```

```
auto ptr = NULL;
auto ptr2 = nullptr;

cout << typeid(ptr).name();
cout << typeid(ptr2).name();
```

various new C++11 std:: functions take only nullptr

legacy C code may not work with nullptr!

https://en.cppreference.com/w/cpp/language/typeid
https://en.cppreference.com/w/cpp/types/type_info
https://en.cppreference.com/w/cpp/types/nullptr_t

Classes that contain dynamic memory

- Or any dynamic resource (file, network, …)

```
class bigData
{
  public:

    bigData();
    bigData(int data);
    ~bigData();
    bigData(const bigData &rcData);

    // bigData& operator=(const bigData &rcData);

    // you cannot declare both operator= at the same time!
    bigData& operator=(bigData cData); //force copy constructor to be called


  private:
    int *mpHugeData = nullptr;
}
```

copy-and-swap

```
bigData::bigData(const bigData &rcData)
{
 // let’s assume this is written and works correctly
}


bigData& bigData::operator=(bigData cData)
{
  // what happens when cBigData1 = cBigData2; is executed?









  // what happens when operator= terminates?

}
```

- What gets called? How many objects are created?

```
bigData cBigData (1);
bigData cBigData2 (cBigData);
bigData cBigData3 ;


cBigData3 = cBigData2;
```

ctor, cctor, dtor?

Smart Pointers

- C++11 wrapper classes to manage pointers
  - RAII for pointers
  - <memory>
- unique_ptr
- shared_ptr
- weak_ptr

https://msdn.microsoft.com/en-us/library/hh279674.aspx

http://umich.edu/~eecs381/handouts/C++11_smart_ptrs.pdf

raw pointer

- int *pRawPointer;
- Issues:

std::unique_ptr

```
std::unique_ptr<bigData> pBigData(new bigData(42));

auto pBigDataAgain(std::make_unique<bigData>(bigData(41)));
```

https://en.cppreference.com/w/cpp/memory/unique_ptr

std::shared_ptr

```
std::shared_ptr<bigData> pBigDataShared(new bigData(7));

auto pBigDataSecondShared = pBigDataShared;

auto pBigDataSharedAgain (std::make_shared<bigData> (bigData (41)));
```

https://en.cppreference.com/w/cpp/memory/shared_ptr

Lab - use raw pointers

- You must write your own string class, backed by a dynamic char array.
- The string class must implement the interface provided.
  - PacString.h
- Use the provided test driver. main.cpp
  - make sure there are no memory leaks
  - determine how many times each of the following is called by the test driver:
    - default constructor
    - constructor (const char *)
    - copy constructor
    - destructor

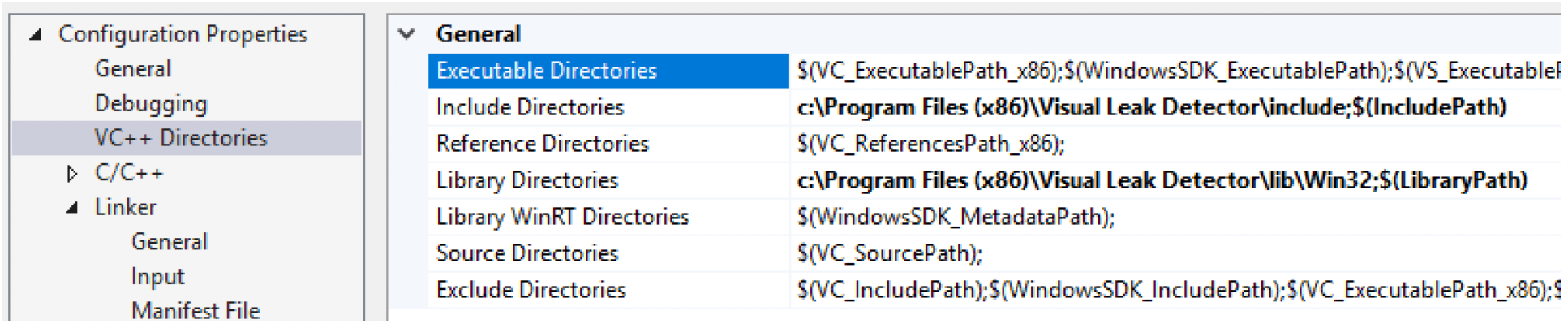
Visual Leak Detector

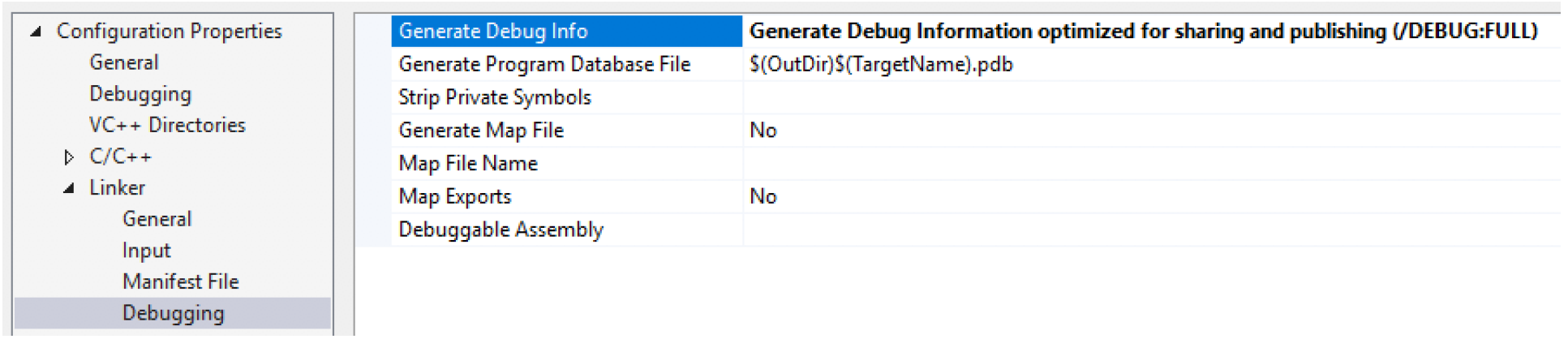
- https://kinddragon.github.io/vld/

- Project Properties | Linker | Debugging | Generate Debug Info[1]
  - "Generate Debug Information optimized for sharing and publishing (/DEBUG:FULL)"

[1]https://stackoverflow.com/questions/44708137/visual-leak-detector-with-visual-studio-2017-no-source-code-line-numbers/44724869#44724869

Using VLD


Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
General
Executable Directories
$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutableP
Include Directories
c:\Program Files (x86)\Visual Leak Detector\include;$(IncludePath)
Reference Directories
$(VC_ReferencesPath_x86);
Library Directories
c:\Program Files (x86)\Visual Leak Detector\lib\Win32;$(LibraryPath)
Library WinRT Directories
$(WindowsSDK_MetadataPath);
Source Directories
$(VC_SourcePath);
Exclude Directories
$(VC_IncludePath);$(WindowsSDK_IncludePath);$(VC_ExecutablePath_x86);$


Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
Generate Debug Info
Generate Debug Information optimized for sharing and publishing (/DEBUG:FULL)
Generate Program Database File
$(OutDir)$(TargetName).pdb
Strip Private Symbols
Generate Map File
No
Map File Name
Map Exports
No
Debuggable Assembly

Next time

- Be sure to review the example on page 11 in Shalloway