ECE 3220
Templates and Smart Pointers

Dr. Ekin
Jaired Collins, Gbenga Omotara

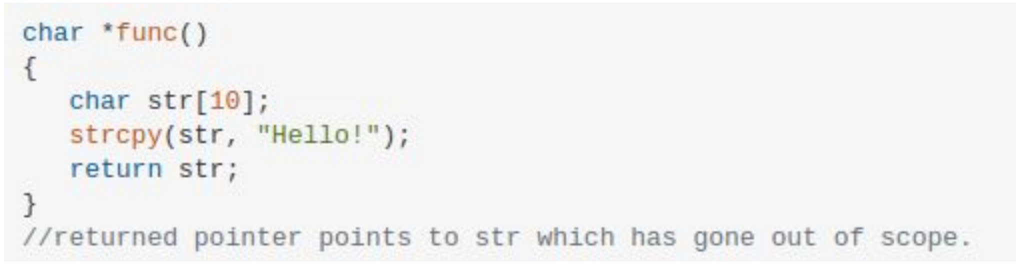
Templates

- Dr. Ekin covered a lot of its importance
- Examples in code

- Templated classes/functions are not classes or functions!
- They are just that, a template, a recipe
- The compiler must see the need for it before it is filled in
- Special care in implementing templated classes and functions is needed
- This may be a hard concept at first, but you’re really just making room for an extra parameter that allows you to specify the type
- Templated code is associated with < >
    - Where have you seen this before?
    - /usr/include/c++/7/vector and bits/stl_vector.h, vector.tcc

What’s wrong with dumb pointers?

- Forget to delete
  - Memory leak
- Returning memory that was already freed
  - Dangling pointer



```
char *func()
{
   char str[10];
   strcpy(str, "Hello!");
   return str;
}
//returned pointer points to str which has gone out of scope.
```

Smart Pointers

- Garbage collection
  - C#, Java, …
  - Does C++ have it?
    - If it does, why use smart pointers? (Useless?)
    - If it doesn't, why use smart pointers? (Missing feature?)

Put scope on pointers

- How do you force a scope, and therefore destruction, on a heap pointer?

1. Encapsulate the pointer in an object
2. Create a destructor that deletes the pointer
    a. When is this used?
3. Overload the -> and * operators
    a. The underlying pointer can now be reached seamlessly

Built-in Smart Pointers

- std::unique_ptr
  - Allows exactly one owner of the underlying pointer
    - No copying or sharing through code
  - Your ‘go-to’, hopefully not goto though
- std::shared_ptr
  - Allows multiple owners, use if the pointer needs to be shared
  - Counts the number of references, once it hits 0 or all shared_ptr owners go out of scope, it deletes
- std::weak_ptr
  - Special-case to be used with shared_ptr
  - Used to provide looking, if needed, and not contributing to a reference count
  - Required in some cases to break circular references

Example 1

```
void UseRawPointer()
{
    // Using a raw pointer -- not recommended.
    Song* pSong = new Song(L"Nothing on You", L"Bruno Mars");

    // Use pSong...

    // Don't forget to delete!
    delete pSong;
}


void UseSmartPointer()
{
    // Declare a smart pointer on stack and pass it the raw pointer.
    unique_ptr<Song> song2(new Song(L"Nothing on You", L"Bruno Mars"));

    // Use song2...
    wstring s = song2->duration_;
    //...

} // song2 is deleted automatically here.
```

Example from https://docs.microsoft.com/en-us/cpp/cpp/smart-pointers-modern-cpp?view=msvc-160

Example 2

```
class LargeObject
{
public:
    void DoSomething(){}
};

void ProcessLargeObject(const LargeObject& lo){}
void SmartPointerDemo()
{
    // Create the object and pass it to a smart pointer
    std::unique_ptr<LargeObject> pLarge(new LargeObject());

    //Call a method on the object
    pLarge->DoSomething();

    // Pass a reference to a method.
    ProcessLargeObject(*pLarge);

} //pLarge is deleted automatically when function block goes out of scope.
```

Example from https://docs.microsoft.com/en-us/cpp/cpp/smart-pointers-modern-cpp?view=msvc-160

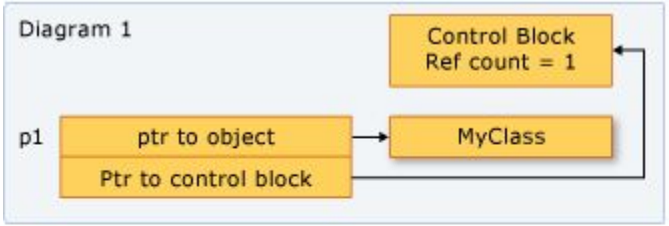
Example 3

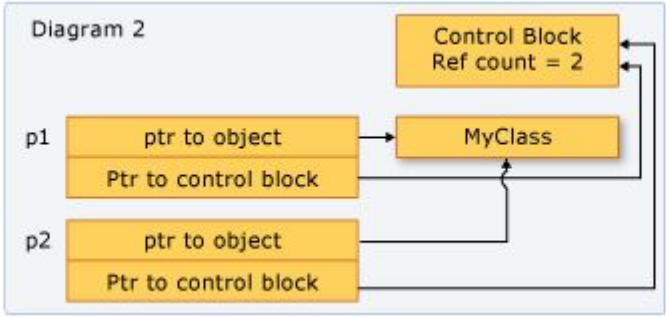
```
// Use make_shared function when possible.
auto sp1 = make_shared<Song>(L"The Beatles", L"Im Happy Just to Dance With You");

// Ok, but slightly less efficient.
// Note: Using new expression as constructor argument
// creates no named variable for other code to access.
shared_ptr<Song> sp2(new Song(L"Lady Gaga", L"Just Dance"));

//Initialize with copy constructor. Increments ref count.
auto sp3(sp2);

//Initialize via assignment. Increments ref count.
auto sp4 = sp2;
```





Example from https://docs.microsoft.com/en-us/cpp/cpp/smart-pointers-modern-cpp?view=msvc-160