**C++/CLI**

1. **Blittable & Non-Blittable Types**

C# and C++ has data types related issues - C# cannot call char, struct, objects,… 🡪 Non-Blittable types

* e.g, In C#, char is 16bits, in C++, char is 8bits

Blittable 🡪 System.Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64, IntPtr, UIntPtr, Single, Double (same in C# and C++)

1. **C++/CLI** - Common Language Infrastructure (**Managed**)

DLLs that wraps native C++ code (binary routines) and make it **accessible from .NET programs**

Can access both C++ and .NET objects

\*\*In project: A Wrapper class is created, which has a pointer to a “**new**” C++ engine object and a bunch of calculation methods. These methods takes in C# arguments and invokes C++ functions of the engine

* Create a Wrapper object in C# and invokes its methods for calculations

**Syntax**:

* **ref** = added before class definition to designate for CLI/CLR
* **^** = managed version of \* (unmanaged) for pointers (CLI heap)
  + (.NET) System::String hi == (C++/CLI) System::String^ hi (String is reference type in C#)
  + MUST add for any types that were reference type in C#
  + HOWEVER, their memory allocation now is similar to C++ (not automatically reference)
* **gcnew** = Creates a reference to a .NET object (managed)
* **!Class()** = Finalizers (in case the destructor of a class isn’t called - forgot to “delete”)

\*\*gcnew is creating an object in C# (Heap), while doing “MyClass obj” in C++/CLI (Stack, Value)

**Arrays**

In C++, an array is just blocks of memory, but in .NET, an array is an **object**

In C#, arrays were of reference type

array<String^>^ str\_array = { 1, 2, 3, 4 }

array<String^>^ str\_array = gcnew array<String^>(4);

Array of arrays:

array<array<String^>^>^ str\_array = gcnew array<String^>(3); // <- 3 rows

str\_array[0, 0]

**pin\_ptr**

* A pinning pointer is an interior pointer that prevents the object pointed to from moving on the garbage-collected heap. That is, the value of a pinning pointer is not changed by the common language runtime.
* This is **required** when you pass the address of a managed class to an unmanaged function so that the address will not change unexpectedly during resolution of the unmanaged function call.

[cli::]pin\_ptr<cv\_qualifiertype> var = &initializer;



A screenshot of a computer

Description automatically generated with low confidence *(native\_function unmanaged)*

**msclr::interop::marshal\_as**

Defined in #include <msclr\marshal\_cppstd.h> header file

The marshal\_as function uses a technique called "marshaling" to convert between managed and unmanaged types.

A black background with white text

Description automatically generated with low confidence

e.g: std::string unmanagedid = msclr::interop::marshal\_as<std::string>(id);