LEWG, SG14: D0040R0   
22-9-2015   
Scott Wardle, Roberto Parolin  
[swardle@gmail.com](mailto:swardle@gmail.com), [robertoparolin@gmail.com](mailto:robertoparolin@gmail.com)

**Comparing Virtual Methods**

**I. Summary**

It would be useful to compare if an instance of a class has a particular function in its virtual table.

**II. Motivation**

Traditional entity systems in c++ are used often in games. You have a list of entity base classes some of which are overridden and these data driven and not known at compile time. The traditional solution for updating the entities is looping over a vector of the entity base class and calling its update function. The virtual function overhead could be expensive in some cases so sorting the entity vector by if the entity base class update method has been overridden would have the performance advantages of better branch prediction and icache reuse.

We would use this to scan though vectors of entities looking for overridden update functions. We would move these overridden objects out of the fast path and have one function that updated many non-overridden entities. Then update each overridden entity slowly afterwards.

## III. Additional work

In addition being able to use the virtual table as type id, it might be useful in some cases to sort array of entity by type not just by function. If the entities have multiple update functions it may be beneficial to sort by a type id rather than the update methods. However it is often the case that sorting by function would lead to better performance if not many entity types change the default update function.

We need to review <http://en.cppreference.com/w/cpp/types> type\_index or hash\_code most games don't use rtti however. The memory costs of rtti are not clear to me. Maybe we can only add rtti information to classes that use get a type id from.

**III. Discussion**

Roughly I want to do things like the following C# code can.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace cs\_func\_sort

{

class A

{

virtual public void Foo()

{

Console.WriteLine("A::Foo");

}

virtual public void Name()

{

Console.Write("Class A ");

}

}

class B : A

{

override public void Foo()

{

Console.WriteLine("B::Foo");

}

override public void Name()

{

Console.Write("Class B ");

}

}

class C : B

{

override public void Name()

{

Console.Write("Class C ");

}

}

delegate void fptrDelegate();

class Program

{

static void Main(string[] args)

{

List<A> list = new List<A>();

list.Add(new A());

list.Add(new C());

list.Add(new B());

list.Add(new A());

list.Add(new C());

Console.WriteLine("Print the list unsorted");

foreach (A cur in list)

{

cur.Name();

cur.Foo();

}

Console.WriteLine("Print the list sorted");

list.Sort(delegate (A itemA, A itemB)

{

fptrDelegate afoo = itemA.Foo;

fptrDelegate bfoo = itemB.Foo;

return afoo.Method.GetHashCode() - bfoo.Method.GetHashCode();

});

foreach (A cur in list)

{

cur.Name();

cur.Foo();

}

Console.WriteLine("Find all of the version with B");

// Can't think of a good way to get rid of creating this object in C# in C++

// in C++ it would be better if we didn’t have to create a instance of the class

// to make a pointer to function like the example below in yellow.

A testA = new B();

fptrDelegate testfoo = testA.Foo;

foreach (A cur in list)

{

fptrDelegate curfoo = cur.Foo;

if( testfoo.Method.Equals(curfoo.Method) )

{

// call B::Foo AND C::Foo!!

cur.Name();

cur.Foo();

}

}

}

}

}

**IV. Proposed Text**