Subject: [SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Fri, 13 Sep 2019 14:32:37 GMT
View Forum Message <> Reply to Message

Hello,

Is it possible to do something like that :

```
#include <Core/Core.h>

using namespace Upp;

class A : public Moveable<A>{
 public:
 A(){}
 virtual void Hello(){
  Cout() << "Hello from A" <<"\n";
 }
};

class B :public A, public Moveable<B>{
 public:
 B(){}
 void Hello(){
  Cout() << "Hello from B" <<"\n";
 }
};

CONSOLE_APP_MAIN
{
 Vector<A> myVector;

 static_cast<B&>(myVector.Add()).Hello();
 //Looking for "Hello from B"
}
```

Without using Vector of ptr or reference ?

Thanks in advance
Best regard.

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Fri, 13 Sep 2019 19:55:44 GMT
View Forum Message <> Reply to Message

Hello Xemuth,

Take a look at the class Array. It is capable of doing this.

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Fri, 13 Sep 2019 22:29:07 GMT
View Forum Message <> Reply to Message

Class A (and B) should have a virtual destructor in this case.

Subject: Re: Vector of object: cast to inherited class
Posted by Xemuth on Sat, 14 Sep 2019 11:41:07 GMT
View Forum Message <> Reply to Message

Hello Novo, Thanks for you respons.

I have take a look at Array documentation and it say :
Quote:Array can be also used to store polymorphic elements - stored elements could be derived from T. To store such elements, you pass a pointer to an element previously created on the heap. Still, Array takes over ownership of such element (it e.g. deletes it when appropriate). You can also use this method to create an Array of elements that do not have either pick, deep copy constructor nor default constructor.

"To store such elements, you pass a pointer to an element previously created on the heap"
It mean I must use an Array<A*> ? Their is no way to do something like the exemple bellow without having to work with ptr ?

Thanks in advance, have good day.

```
#include <Core/Core.h>

using namespace Upp;

class A {
 public:
 A(){}
 virtual ~A(){}
 virtual void Hello(){
  Cout() << "Hello from A" <<"\n";
 }
};

class B :public A{
 public:
 B(){}
 ~B(){}
 void Hello(){
```

```
  Cout() << "Hello from B" <<"\n";
 }
};

CONSOLE_APP_MAIN
{
 Array<A> myArray;
 B b;
 static_cast<B&>(myArray.Add(b)).Hello(); //"Hello from A"
 ((B&)myArray.Add(b)).Hello(); //"Hello from A"
 //Looking for "Hello from B" without using Array<A*>

 Array<A*> myArray2;
 myArray2.Add(&b)->Hello(); //Working properly but using ptr
}
```

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Sat, 14 Sep 2019 19:39:26 GMT
View Forum Message <> Reply to Message

Check how code below works.
A hint: I'm not using a keyword class. I'm using struct instead. This makes code shorter and cleaner.

```
struct A {
 A() {}
 virtual ~A() {}
 virtual void Hello() const {
  Cout() << "Hello from A" << EOL;
 }
};

struct B : A {
 B() {}
 ~B() {}
 void Hello() const {
  Cout() << "Hello from B" << EOL;
 }
};

CONSOLE_APP_MAIN
{
 Array<A> arrA;
 arrA.Create<B>();
 arrA.Add(new B);
 arrA.Add();
 for (const A& a: arrA)
  a.Hello();
```

}

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Sat, 14 Sep 2019 19:45:42 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Sat, 14 September 2019 07:41
It mean I must use an Array<A*> ? Their is no way to do something like the exemple bellow without having to work with ptr ?

Array<A> is similar to Vector<A*>.
The only difference is that it (Array<A>) owns data. This means that Array<A> in its destructor will call delete on each pointer it stores, so, you cannot store pointers to objects on stack in an Array.

Subject: Re: Vector of object: cast to inherited class
Posted by Xemuth on Sun, 15 Sep 2019 10:52:48 GMT
View Forum Message <> Reply to Message

Hello Novo, Thanks for your exemple !

Also Thanks for showing me "EOL" value allow me to quickly do a \n !

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Sun, 15 Sep 2019 15:31:20 GMT
View Forum Message <> Reply to Message

No problem.
A quiz for you.
Why the code below is working the way it is working (printing out "Hello from A" instead of "Hello from B")? :)

```
struct A {
 virtual ~A() {
  Hello();
 }
 virtual void Hello() const {
  Cout() << "Hello from A" << EOL;
 }
};

struct B : A {
 void Hello() const {
  Cout() << "Hello from B" << EOL;
```

```
 }
};

CONSOLE_APP_MAIN
{
 Array<A> arrA;
 arrA.Create<B>();
}
```

Subject: Re: Vector of object: cast to inherited class
Posted by Novo on Sun, 15 Sep 2019 15:54:32 GMT
View Forum Message <> Reply to Message

BTW, you do not have to define a default constructor all the time. It is automatically generated by a compiler for you.
You only need to do that when its behavior is different from standard one.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Mon, 16 Sep 2019 07:35:37 GMT
View Forum Message <> Reply to Message

Hello Novo,

Quote:Why the code below is working the way it is working (printing out "Hello from A" instead of "Hello from B")? Smile

That's because we didn't define any destructor on B.

Edit : I just tried to define B destructor and A destructor is still called after B destructor call.
That's not the behaviour I would have imagined but it's quite logique.

Quote:A hint: I'm not using a keyword class. I'm using struct instead. This makes code shorter and cleaner.
Also Except Struct is well aligned in memory and you didn't set accessor flag to public, what's the difference between class and struct ?

Thanks in advance.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Novo on Mon, 16 Sep 2019 14:29:37 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 16 September 2019 03:35Also Except Struct is well aligned in memory and you didn't set accessor flag to public, what's the difference between class and struct ?

Default visibility for class is private.
Default visibility for struct is public.
That includes inheritance.
There is no other difference.
Memory alignment is a completely separate topic.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Novo on Mon, 16 Sep 2019 14:31:48 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 16 September 2019 03:35
Quote:Why the code below is working the way it is working (printing out "Hello from A" instead of "Hello from B")?

That's because we didn't define any destructor on B.

This is not a correct answer.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Mon, 16 Sep 2019 14:47:29 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 16 September 2019 16:31Xemuth wrote on Mon, 16 September 2019 03:35
Quote:Why the code below is working the way it is working (printing out "Hello from A" instead of "Hello from B")?

That's because we didn't define any destructor on B.

This is not a correct answer.

Since B inherite from A, B will call A's destructor right after exection of is own destructor.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Novo on Mon, 16 Sep 2019 15:44:53 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 16 September 2019 10:47Novo wrote on Mon, 16 September 2019 16:31Xemuth wrote on Mon, 16 September 2019 03:35
Quote:Why the code below is working the way it is working (printing out "Hello from A" instead of

"Hello from B")?

That's because we didn't define any destructor on B.

This is not a correct answer.

Since B inherite from A, B will call A's destructor right after exection of is own destructor.
This particular statement is correct, but Hello() is a virtual method and it is supposed to print "Hello from B".

```
B b;
A* a = &b;
a->Hello();
```

This code will print "Hello from B".
Why the code in quiz is working differently?

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Mon, 16 Sep 2019 18:02:28 GMT
View Forum Message <> Reply to Message

A only know is own function ?

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Novo on Mon, 16 Sep 2019 18:09:46 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 16 September 2019 14:02A only know is own function ?
Can you explain what you mean by that step by step?
This quiz is for you to understand how virtual methods work.

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Mon, 16 Sep 2019 18:25:19 GMT
View Forum Message <> Reply to Message

Define Hello() as virtual on A allow us to redefine Hello() on Children but since B destructor will call A destructor at is exit and
A don't know if child have possible redefinition of Hello() he will call is own Hello() declaration. :d

---

Subject: Re: |SOLVED] Vector of object: cast to inherited class

Posted by Novo on Mon, 16 Sep 2019 21:47:50 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 16 September 2019 14:25Define Hello() as virtual on A allow us to redefine Hello() on Children but since B destructor will call A destructor at is exit and
A don't know if child have possible redefinition of Hello() he will call is own Hello() declaration. :d
Still wrong. In case of "a->Hello();" a doesn't know about possible redefinition of Hello() as well, but it still prints "Hello from B".

And destructor of B doesn't call destructor of A. It works in a different way, although the order of calls is the same.

And you do not have to declare Hello() in A as virtual if you want to redefine it in B. Code below will compile.

```
struct A {
 virtual ~A() {
  Hello();
 }
 void Hello() const {
  Cout() << "Hello from A" << EOL;
 }
};

struct B : A {
 void Hello() const {
  Cout() << "Hello from B" << EOL;
 }
};
```

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Tue, 17 Sep 2019 07:55:00 GMT
View Forum Message <> Reply to Message

OK, I got it, If we don't define Hello() as Virtual then if we do this kind of thing :

```
 B b;
 A* a = &b;
 a->Hello(); //Printing "Hello from A"
```

Since Hello() is static defined C++ will only use type of declared variable to know what function he must call.
if "a" was B* then we would have call B::Hello() etc...
However, if we set Hello() as virtual in A and redifine it in B then the function C++ must call will not be define at compilation but during the runtime, for each call of Hello() C++ will test the real type of Object (I don't know how he do it) then call the right definition (based on type of object) of the

function.

B b;
A* a = &b; //here Hello() is virtual in A and redefined in B
a->Hello();//C++ try to know real type of "a" and see it's a B object (or maybe a B* even if I store it in A*) then Printing "Hello from B"

:d :d

Subject: Re: |SOLVED] Vector of object: cast to inherited class
Posted by Xemuth on Tue, 17 Sep 2019 12:11:42 GMT
View Forum Message <> Reply to Message

Fun fact: I had trouble in my application using Array classs it was like inherited class wasn't able to call their destructor instead of calling mother class destructor. Just looked if mother destrutor was virtual and it was not ! Thanks to your quizz I have been able to spot the probleme quickly :lol:
