Subject: Raise Exception instead of ASSERT for container
Posted by benoitc on Tue, 09 Oct 2007 09:03:27 GMT
View Forum Message <> Reply to Message

Hi All,

I think that it will be better (at least for me) to let container raising an exception during out of bound access (like in Java) instead of an assert.

Today we need to check index before any access to array element and thus we cannot easily use double indirection like array[i][j].

What do you think about that? Does that sound stupid?

Regards,
Benoit

Subject: Re: Raise Exception instead of ASSERT for container
Posted by unodgs on Tue, 09 Oct 2007 09:50:07 GMT
View Forum Message <> Reply to Message

Cant you just check bounds before getting the value and throw an exception manually?

Subject: Re: Raise Exception instead of ASSERT for container
Posted by mirek on Tue, 09 Oct 2007 12:46:10 GMT
View Forum Message <> Reply to Message

benoitc wrote on Tue, 09 October 2007 05:03Hi All,

I think that it will be better (at least for me) to let container raising an exception during out of bound access (like in Java) instead of an assert.

I guess the idea is that invalid index is program logic problem - nothing that should be resolved by exception.

Quote:
Today we need to check index before any access to array element and thus we cannot easily use double indirection like array[i][j].

How would exception help with double indirection here? (scratching my head...

Mirek

Subject: Re: Raise Exception instead of ASSERT for container
Posted by tvanriper on Tue, 09 Oct 2007 13:07:22 GMT
View Forum Message <> Reply to Message

I'm not a personal fan of exceptions, myself, unless the reason for the exception is fairly extraordinary.

I guess it's kind of a matter of opinion as to whether or not this sort of thing merits an exception.

If you're thinking like a C programmer, yeah, it merits it. After all, you're potentially accessing unallocated memory (always a no-no).

But if you're thinking like a C++ programmer, it may not. We're working with objects here, who can return other objects. The object returned may hold state that indicates an error (or may itself be indicative of an error).

One need only test the object to ensure its validity before proceeding... no exception required, and the code is extremely straight-forward to read (doesn't potentially jump around to some catch(...) statement).

I suppose, though, one could optionally 'enable' exceptions in the object, in case that form of error handling is for some reason preferable.

Subject: Re: Raise Exception instead of ASSERT for container
Posted by benoitc on Tue, 09 Oct 2007 16:26:01 GMT
View Forum Message <> Reply to Message

The goal of exceptions in C++ is to clearly separate nominal code with exceptional errors management, instead of adding a bunch of test before calling any API just in case an error occurs.

Without exception you need to check first that "i" in inside the bound then get a reference to array[i] and then check than "j" is inside the bound of the array[i] container.

Don't you think that it might be easier to use exception to handle potential error on array access?

The usage of exception is more a matter of taste than anything else, so if you're not convince that it might be useful for that kind of stuff, I'm fine with that.

Regards,
Benoit

Subject: Re: Raise Exception instead of ASSERT for container
Posted by tvanriper on Tue, 09 Oct 2007 18:36:33 GMT
View Forum Message <> Reply to Message

Another alternative (one I suggested in my original response, but perhaps too subtly) to using

exceptions is to simply perform your request for the object within your multi-dimensional array, then test to see that the object is valid. If it is, it was within bounds. If not, you were outside of bounds (the array purposefully gave you a kind of object that is not valid, to give you a hint that you attempted an illegal operation... 'Value' objects have state to indicate error conditions in U++, and are perfect for this kind of thing).

No need to test to see if you're within bounds using this kind of code. Just test to see if the item you requested from the array is valid.

Such code is easier to read, in my opinion, than exceptions, which have a way of disrupting the flow of the code, sometimes in very undesirable ways (e.g. you were compelled to use a 'new' statement, and the exception has now caused your object to become dereferenced).

But, as we both seem to agree, this sort of thing may be a matter of personal preference. I think I've seen other examples of code where exceptions are enabled as an option; perhaps such code is appropriate here.

Subject: Re: Raise Exception instead of ASSERT for container
Posted by mirek on Tue, 09 Oct 2007 21:33:45 GMT
View Forum Message <> Reply to Message

benoitc wrote on Tue, 09 October 2007 12:26The goal of exceptions in C++ is to clearly separate nominal code with exceptional errors management, instead of adding a bunch of test before calling any API just in case an error occurs.

Mostly correct, but "errors" is confusing. Exceptional situation is much better term. Such exceptional situation are intiated by conditions that cannot be predicted by program logic, like not enough space on HD or invalid format of file being read.

Quote:
Without exception you need to check first that "i" in inside the bound then get a reference to array[i] and then check than "j" is inside the bound of the array[i] container.

Incorrect (IMO). The contract of Vector::operator[] has precondition that index is in 0 - GetCount() range.

Quote:
Don't you think that it might be easier to use exception to handle potential error on array access?

Definitely not. Exception are not there to fix programmers errors. At least not in U++ and I believe not even Bjarne/Stepanov/Koenig et al meant such thing to be.

Well, something completely different is if the index would be result of some input, e.g. it woul be in input file. Then using exception to stop parsing of intput file is completely valid.

However, that should be done by parser code. If nothing else, testing the index in each operator[] would make code in some cases as much as 500% slower (as usually compiler is able to optimize the loop to the pointer scan).

Quote:
The usage of exception is more a matter of taste than anything else, so if you're not convince that it might be useful for that kind of stuff, I'm fine with that.

Well, do not get me wrong, I think this is a useful debate.

Plus, you still have not explained how exceptions are going to help with multidimensional arrays....

(BTW, v.At(x).At(y) is method to deal with them in U++).

Mirek

Subject: Re: Raise Exception instead of ASSERT for container
Posted by benoitc on Wed, 10 Oct 2007 08:46:24 GMT
View Forum Message <> Reply to Message

OK, It was just a suggestion, I think we clearly don't have the same perception of the exception usage in C++:

Quote:Mostly correct, but "errors" is confusing. Exceptional situation is much better term. Such exceptional situation are intiated by conditions that cannot be predicted by program logic, like not enough space on HD or invalid format of file being read... Quote:Definitely not. Exception are not there to fix programmers errors. At least not in U++ and I believe not even Bjarne/Stepanov/Koenig et al meant such thing to be.

I'm not that sure about that, because if you refer to C++ bible (Bjarne) in the chapter 14, paragraph 14.1.1 (Alternative Views on Exceptions), quoting the Master
Quote:Bjarne:"Exception is one of those words that means different things to different people"
Here at least we all agree
Quote:Bjarne:The C++ exception-handling mechanism is designed to support handling of errors and other exceptional condition(hence the name)... This mechanism is designed to handle only synchronous exceptions, such as array range checks...

It looks like I'm not that far from the definition of the Master. For me, as soon as you have a function that return directly a value/object/reference and that can potentially fail, you could/should throw an exception to let the user of that API deals with such case and avoiding adding a bunch of test before the call (sometime it is not even possible).

Quote:Plus, you still have not explained how exceptions are going to help with multidimensional arrays...

I thought I did: How to you check nicely that j is a valid index in the v[i][j]? I personally think it is a pain to do if(i>=0&&i<v.GetCount()&& j>=0 && j<v[i].GetCount()) (imagine in the case of 3 or 4 dimension) each time I want to access the array, I would prefer that to be handled by the vector class itself through exception.

BTW, I think that v.At(y) if a little bit different than
v[y], because in my case I don't want to increase the size, I want to access an already allocated an populated array, but I use a index (retrieve from a editable file) that can be wrong (<0 or >GetCount).

As I said, It was just a suggestion based on my perception of the exception usage in C++ or in Java. If you don't want to use that in U++, I cannot blame you, your code is so excellent than I'm fine if only one minor thing does not please me in U++.

Regards,
Benoit

Subject: Re: Raise Exception instead of ASSERT for container
Posted by mirek on Wed, 10 Oct 2007 10:30:34 GMT
View Forum Message <> Reply to Message

benoitc wrote on Wed, 10 October 2007 04:46
Quote:Bjarne:The C++ exception-handling mechanism is designed to support handling of errors and other exceptional condition(hence the name)... This mechanism is designed to handle only synchronous exceptions, such as array range checks...

It looks like I'm not that far from the definition of the Master.

Well, I was rather refering to the fact that STL does not check indices as well... (unless you use vector::at).

I think what is the Master up to here is rather the fact that exceptions are not to be used to handle e.g. interrupts... OTOH, nice contraargument

Quote:
Quote:Plus, you still have not explained how exceptions are going to help with multidimensional arrays...

I thought I did: How to you check nicely that j is a valid index in the v[i][j]? I personally think it is a pain to do if(i>=0&&i<v.GetCount()&& j>=0 && j<v[i].GetCount())

OK