

Forum:[Hack Forums](#)

Thread: [Advanced Snake Game](#)

programtesting28 - July 19, 2017, 6:57 am

Hello HF,

Today I wanna share my collage project.

This is Advanced Snake Game coded with OOP concept.

There are an AI BOT with A* Algorithm implementation.

You can learn many basic of C++ from this project.

This is some screenshot:

Download Link:

<https://www.dropbox.com/s/gdujejrtw638npo/Project%20Snake.zip?dl=0>

Note: there is solution file that can be opened by visual studio, but there are .cpp files too that you can open with any text editor.

Dostoevsky - July 19, 2017, 10:09 am

I have some points of advice.

1) Stop passing every STL class by-value. So many copies are going to be made. It's weird that you noticeably pass your own types by reference yet every instance of an STL class you pass by value. There's only really 3 prominent cases where optimisation can be performed there:

1.1) If you pass an rvalue expression such as a conversion construction, the copy will be elided.

1.2) Similarly, if you're returning by value, the same optimisation will happen implicitly for you (under the other name of RVO - return value optimisation).

1.3) If you explicitly cast an lvalue to an rvalue when passing it w/ `std::move()`. Then, if possible, it will attempt to move construct.

You should keep this in mind because there's a lot of cases where (even though a few of the classes you use wouldn't benefit much from being able to move; since they're mostly POD classes - you're not gonna move primitive members) - you should still be aware that you're inadvertently throwing away special member generation when you explicitly declare your destructor (which you do).

I make these points because you actually `push_back` into a `std::vector` with rvalue construction on one of your own types. Despite the fact being able to move the class in question (I think it was the `Wall` class) wouldn't prove too beneficial (for the POD point I spoke about), you should be aware that, as far as I'm aware, you threw away the move constructor special member when you explicitly declared your destructor in the parent class of `Wall`. As I said, though, despite not being very beneficial in this case, you should at least be more aware of it than you seem to be.

I find a lot of people read EMC++ and think passing by value will always produce optimisations, it will.. but only for the points I mentioned. For example, some people think a suitable solution in their mutators is to pass by value and then explicitly attempt to move it upon assignment to their private member (to work around ambiguity and the absolute cancer that would be making your mutators template methods that take universal references so that you can pass both lvalues and rvalues and rely on `std::forward` to resolve the correct move/copy assignment depending on what is passed). This is actually, in my opinion, a very stupid thing to do (as it costs one copy regardless - apart from

the possible by-value optimisations that could be considered).

tl;dr - stop passing STL classes by value every time.

NOTE: A fun thing you could do is go ahead and steal Howard Hinnant's allocator boilerplate and track move constructor and copy constructor invocations when used with STL classes (keep in mind: although compilers follow a specification, some overcompensate and will provide very different outputs when it comes to implicit optimisation). This will hopefully illustrate how ludicrous what you're doing is.

2) I personally consider the fact you return a non-const reference to your private `Coordinate` member in your `Food` class to be a severe violation of encapsulation. It's somewhat subjective though, I personally think providing adequate mutators and accessors would have sufficed (unless your intention was to have the co-ordinates of food instances to be immutable - if that was the case, you'd have it return a const reference and then you'd also be able to make the method itself const - see point 3). I haven't fully read the code deeply so I can't comment much on my assumptions regarding your intentions.

3) Your const-correctness is quite inconsistent. A lot of your accessor methods are const but a lot aren't. Inconsistent.

4) I have a few other petty (I mean petty as fuck) notes but I'll leave them since I haven't bothered to read the entire source code fully (thus my assumptions about what you intended to do are largely misguided).

Sorry for the huge rant in my first point. It's just important because a lot of people (especially on HF) seem to be blissfully unaware of how STL classes should be used (passed around).

programtesting28 - July 20, 2017, 12:22 am

(Jul 19, 2017 10:09 AM)Dostoevsky Wrote: [\[CITE\]](#) I have some points of advice.

1) Stop passing every STL class by-value. So many copies are going to be made. It's weird that you noticeably pass your own types by reference yet every instance of an STL class you pass by value. There's only really 3 prominent cases where optimisation can be performed there:

1.1) If you pass an rvalue expression such as a conversion construction, the copy will be elided.

1.2) Similarly, if you're returning by value, the same optimisation will happen implicitly for you (under the other name of RVO - return value optimisation).

1.3) If you explicitly cast an lvalue to an rvalue when passing it w/ `std::move()`. Then, if possible, it will attempt to move construct.

You should keep this in mind because there's a lot of cases where (even though a few of the classes you use wouldn't benefit much from being able to move; since they're mostly POD classes - you're not gonna move primitive members) - you should still be aware that you're inadvertently throwing away special member generation when you explicitly declare your destructor (which you do).

I make these points because you actually push_back into a `std::vector` with rvalue construction on one of your own types. Despite the fact being able to move the class in question (I think it was the `Wall` class) wouldn't prove too beneficial (for the POD point I spoke about), you should be aware that, as far as I'm aware, you threw away the move constructor special member when you explicitly declared your destructor in the parent class of `Wall`. As I said, though, despite not being very beneficial in this case, you should at least be more aware of it than you seem to be.

I find a lot of people read EMC++ and think passing by value will always produce optimisations, it will.. but only for the points I mentioned. For example, some people think a suitable solution in their mutators is to pass by value and then explicitly attempt to move it upon assignment to their private member (to work around ambiguity and the absolute cancer that would be making your mutators template methods that take universal references so that you can pass both lvalues and rvalues and

rely on `std::forward` to resolve the correct move/copy assignment depending on what is passed). This is actually, in my opinion, a very stupid thing to do (as it costs one copy regardless - apart from the possible by-value optimisations that could be considered).

tl;dr - stop passing STL classes by value every time.

NOTE: A fun thing you could do is go ahead and steal Howard Hinnant's allocator boilerplate and track move constructor and copy constructor invocations when used with STL classes (keep in mind: although compilers follow a specification, some overcompensate and will provide very different outputs when it comes to implicit optimisation). This will hopefully illustrate how ludicrous what you're doing is.

2) I personally consider the fact you return a non-const reference to your private `Coordinate` member in your `Food` class to be a severe violation of encapsulation. It's somewhat subjective though, I personally think providing adequate mutators and accessors would have sufficed (unless your intention was to have the co-ordinates of food instances to be immutable - if that was the case, you'd have it return a const reference and then you'd also be able to make the method itself const - see point 3). I haven't fully read the code deeply so I can't comment much on my assumptions regarding your intentions.

3) Your const-correctness is quite inconsistent. A lot of your accessor methods are const but a lot aren't. Inconsistent.

4) I have a few other petty (I mean petty as fuck) notes but I'll leave them since I haven't bothered to read the entire source code fully (thus my assumptions about what you intended to do are largely misguided).

Sorry for the huge rant in my first point. It's just important because a lot of people (especially on HF) seem to be blissfully unaware of how STL classes should be used (passed around).

Wow, thanks for your advice. I'm still new at C++, I learn alot from your advice, thank you!