Forum:Hack Forums

Thread: Classes in C

ctrlaltdestroy_ - *October 16, 2013, 7:34 pm*

I'll be going over a simple implementation of object orientation in C, and by the end of this tutorial you'll have set up a simple framework to set up and use simple classes in C. Note: this tutorial will use pointers, function pointers, and etc so don't bother reading if you can't program proficiently in C.

All code is c99 and was compiled under mingw.

Understanding classes and object oriented programming

In object oriented programming, data is private and should not be manipulated directly. Instead, we use member functions that can access the private data. Why might we want to do this? It simplifies and streamlines development and is a more abstract and higher level way of programming. It's "the way" at the moment, with many high schools offering only OOP languages (high schools in my area at least).

In a class, each instance has its own private data set, but uses a table of function pointers for member functions (it does not have its own functions, instead it shares them).

For our implementation, polymorphism and other advanced topics will not be covered. Instead, we'll make a simple String class that is guaranteed to be null terminated (safe).

Since data opaqueness is essential in object oriented programming, make sure to typedef all exported struct pointers to void*.

Setup

For our classes, we're going to make generic allocators, so an instance of any class can be created using the same function. To do this, we'll need to pass a uniform descriptor struct (typedef'd to a void pointer), the same one for each class. Each class will also need to have a pointer to this descriptor struct as its first member, so we can call its constructor/destructor functions without knowing anything else about the class. The second member should be a pointer to the private member block. We also need to know the size of the struct, and size of its private members.

Here's my implementation:
Code typedef void (*ctor)(void* self, va_list* init);
typedef void (*dtor)(void* self);

```
typedef struct Class_base {
    ctor constructor;
    dtor destructor;
    size_t cbClass;
```

```
    size_t cbPriv;
} Class_s;

static const Class_s String_class = {
    String_ctor,
    String_dtor,
    sizeof(String_s),
    sizeof(String_priv)
};

typedef struct String_base {
    void *Class_type,
          *PrivateObjects;
//... member functions
} String_s;
```

And then we will need to pass the class base to the allocator, along with any other initializers.

Prototype for the deallocator and my implementation of an allocator:
Code void del_class(void* _class_object);

```
void* new_class(const void* _class_type, ...) {

    // remove anonymous type
    Class_s* pClass_base = (Class_s*)_class_type;

    // allocate class object
    void* pObj = (void*)calloc(1, pClass_base->cbClass);
    assert_err(pObj, ERR_ALLOC_MEM);

    // set first member as class base
    *(Class_s**)pObj = pClass_base;

    // call appropriate constructor
    if (pClass_base->constructor) {
        va_list vaArgs = NULL;

        // use initializers
        va_start(vaArgs, _class_type);
        pClass_base->constructor(pObj, &vaArgs);
        va_end(vaArgs);
    }

    return pObj;
}
```

Inside the allocator, _class_type will need to be casted to a Class_s pointer, and you will need to allocate an object of cbClass. The first member should be set as a pointer to the _class_type. If there is a constructor, then call that passing the object and the args. The constructor should allocate memory for its private objects.

Inside the deallocator, _class_object will be checked for a destructor, which will free its own private objects. Then, the object should be released.

Constructor/Deconstructor

Since we don't have a __thiscall convention in C, we have to pass the first parameter as this in each member function (I use self to maintain C++ compatibility and because I use a C++ IDE).

Inside the constructor, we will first allocate memory for our private variables (remember that the first member of the object contains a pointer to the _class_type, which has the size of the private data). The second member of the class should always be a pointer to the private data.

The member function pointers will also need to be setup. In the String_s struct, function pointers to each member function should be assigned.

Here's a sample string constructor (remember constructors and deconstructors don't return values):
Code void String_ctor(void* self, va_list* init) {

```
    // remove anonymous types
    String_s* sSelf = (String_s*)self;
    Class_s* pClass_base = (Class_s*)sSelf->Reserved[0];

    // allocate memory for private members
    sSelf->Reserved[1] = calloc(1, pClass_base->cbPriv);
    assert_err(sSelf->Reserved[1], ERR_ALLOC_MEM);

    // initialize exported from class
    sSelf->duplicate = pClass_base->duplicate;

    // initialize rest of function table
    sSelf->c_str = String_c_str;
    sSelf->set = String_set;

    // get initializer and copy
    const TCHAR* tzInit = va_arg(*init, const TCHAR*);
    sSelf->set(self, tzInit);
}
```

And the deconstructor should release the allocated memory for the private objects. This one is simple, so you should be able to implement it yourself.

Member functions

Now we'll finish our simple string class, now that we've got our class system set up. To implement a set function, allocate a buffer _tcslen(tzInitializer) + 1 TCHARs long and set the last one to 0.

To make it more efficient, I decided to allocate memory in blocks of 32, with the max size of the string being 31 to guarantee a null terminator. Then, if I want to concatenate a string, and the length is less than max size, then I can copy it without having to realloc. It also will not allocate if the initializer is smaller than max size, and within a block of it.

For the get function, I decided to go with std::string style and use c_str, which simply returns a const TCHAR pointer to the private member.

Duplicate should simply return a new_class that is initialized to the same c_str as self.

Making it usable

And now to put it together, you can should export the needed class types as void*s.

Code static const void* String_type = (const void*)&String_class;
typedef String_s* String;
And to use it:

Code String s = (String)new_class(String_type, _T("sample"));
    printf("%s", s->c_str(s));

Appendix

My implementation of a string class: DOWNLOAD

What's next? You can make essentially any class with this, very simply following the given template. Maybe I'll make a follow up tutorial with polymorphism and static class functions (c#/java style) next.

vitafortis - *October 17, 2013, 3:57 am*
C is Procedural, C++ is Object Oriented. Are you sure you know what you're doing?

ctrlaltdestroy_ - *October 17, 2013, 6:54 am*
(Oct 17, 2013 03:57 AM)vitafortis Wrote: *[CITE]* C is Procedural, C++ is Object Oriented. Are you sure you know what you're doing?That doesn't mean that you can't make an object oriented system in C. Thinkin outside da box mayne.

Also, I just have been using C by convention for a while, my personal preference. I don't have the time or patience to learn the C++ stdlib/usage, and C works for me :) Also C had variadic macros 12 years faster than C++.

Haha this was just an example to show that it can be done in < 100 lines, quite easily.

Mouse? - *October 17, 2013, 8:44 am*
Don't stray from the 'rules' of the languages dammit D<. You can think outside of the box for your program ideas, not the set customs of the language :C

vitafortis - *October 17, 2013, 8:49 am*
(Oct 17, 2013 06:54 AM)ctrlaltdestroy_ Wrote: *[CITE]* That doesn't mean that you can't make an object oriented system in C. Thinkin outside da box mayne.
But why in the world would you want to? Just use C++ or some other OOP language.

And as Rodaxoleaux said, you think this is a Mother Fucking game?

g4143 - *October 17, 2013, 3:40 pm*

Nice tutorial. I did something like this a while ago but it wasn't this detailed.

ctrlaltdestroy_ - *October 17, 2013, 7:45 pm*

(Oct 17, 2013 08:49 AM)vitafortis Wrote: [CITE] But why in the world would you want to? Just use C++ or some other OOP language.

And as Rodaxoleaux said, you think this is a Mother Fucking game?C is just what I do, haha this is more about convenience than turning C into an object oriented language. Simple example was making a String class that is always zero terminated.

It also exposes a lot of the lower level parts that C++ implements in each class. Actually, it's more or less the same without the syntactic sugar of a superclass (one that doesn't inherit from anything).

[img]http://static.fjcdn.com/comments/What+if+I+told+you+_f2a925c320fcf5b865330a00214e421e.jpg[/img]

What if I told you this was a game? ?

vitafortis - *October 18, 2013, 5:17 am*

(Oct 17, 2013 07:45 PM)ctrlaltdestroy_ Wrote: [CITE] C is just what I do, haha this is more about convenience than turning C into an object oriented language. Simple example was making a String class that is always zero terminated.

It also exposes a lot of the lower level parts that C++ implements in each class. Actually, it's more or less the same without the syntactic sugar of a superclass (one that doesn't inherit from anything).

[img]http://static.fjcdn.com/comments/What+if+I+told+you+_f2a925c320fcf5b865330a00214e421e.jpg[/img]

What if I told you this was a game? ?
While I can understand the premise of what you're doing, is there any great benefit of doing it?

And if it's a game, I'm obviously losing.

g4143 - *October 18, 2013, 5:33 am*

(Oct 18, 2013 05:17 AM)vitafortis Wrote: [CITE] While I can understand the premise of what you're doing, is there any great benefit of doing it?

And if it's a game, I'm obviously losing.
Its a great exercise. It allows you to create a simple OOP environment which shows you how most things bolted together.

vitafortis - *October 18, 2013, 5:37 am*

(Oct 18, 2013 05:33 AM)g4143 Wrote: [CITE] Its a great exercise. It allows you to create a simple OOP environment which shows you how most things bolted together.
I can understand for practice, but I meant outside of it, that's all.

g4143 - *October 18, 2013, 5:46 am*

(Oct 18, 2013 05:37 AM)vitafortis Wrote: [CITE] I can understand for practice, but I meant outside of it, that's all.
Most people don't understand how C++ achieves its OOP magic because its hidden in the

compiling process so the above program is a great example.

vitafortis - *October 18, 2013, 5:50 am*

(Oct 18, 2013 05:46 AM)g4143 Wrote: [CITE] Most people don't understand how C++ achieves its OOP magic because its hidden in the compiling process so the above program is a great example.
Fair enough. I can't argue with that logic. Too many people skip learning the stuff behind the scenes.

Irritiable - *October 18, 2013, 11:12 am*

(Oct 18, 2013 05:50 AM)vitafortis Wrote: [CITE] Fair enough. I can't argue with that logic. Too many people skip learning the stuff behind the scenes.
Dude, what the fuck?
Why would ANYONE want to skip virtual table pointers? That magic is the SHIT!

Being able to make my own customized versi-- Ehh.... I'll just use C++.
I win.

ctrlaltdestroy_ - *October 18, 2013, 1:56 pm*

(Oct 18, 2013 11:12 AM)Irritiable Wrote: [CITE] Dude, what the fuck?
Why would ANYONE want to skip virtual table pointers? That magic is the SHIT!

Being able to make my own customized versi-- Ehh.... I'll just use C++.
I win.
I don't know why you two are arguing, you both use C++, and are apparently happy with it. I use C for most tasks, and don't see myself changing my approach to those domain-specific tasks to C++ in the near future. While I didn't cover polymorphism, it's basically the same, just a redirection to another function by the base class. You don't need to know how it works, sort of like you don't need to understand how C++ conventions in code are converted to assembly; but it's educational (for the children!).

Irritiable I specifically remember you harping on the usage of ++i instead of i++ in code due to the way it's compiled as a fetch-then-increment vs increment. Your argument was that while compilers will most likely produce the same result if either is used in a loop with optimization, you should know why it might instead do differently (as per the standard). How is this different?

I'm getting sidetracked though - point being this was a C thread detailing the implementation in C - I didn't come here to be converted to a mormon C++, and the implementation was C. Objects in C aren't new, look at the Linux kernel.

Irritiable - *October 18, 2013, 2:53 pm*

(Oct 18, 2013 01:56 PM)ctrlaltdestroy_ Wrote: [CITE] I don't know why you two are arguing
Hahahaha. You hear that? He thinks we're arguing. ?