How would one write object-oriented code in C? [closed]

Asked 16 years ago Modified 4 years, 11 months ago

Viewed 404k times

547

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 8 years ago.

Improve this question

What are some ways to write object-oriented code in C? Especially with regard to polymorphism.

See also this Stack Overflow question *Object-orientation in C*.

c oop object

Share

Improve this question

Follow

edited Dec 30, 2019 at 23:15

community wiki
10 revs, 6 users 26%
Peter Mortensen

1 Object Oriented Programming in C by Laurent Deniau – John Mar 31, 2009 at 17:42

33 @Camilo Martin: I intentionally asked *can* not *should*. I'm not actually interested in using OOP in C. However, by seeing OO solutions in C, I/we stand to learn more about the limits and/or flexibility of C and also about creative ways to implement and use polymorphism. – Dinah Nov 24, 2010 at 16:17

7 OO is just a pattern. Check here, it can even be done in **.bat Files:** dirk.rave.org/chap9.txt (any pattern can be applied to any programming language if you are interested enough, I think). This is good food for thought, though. And probably a lot can be learnt from applying such patterns we take for granted on languages that don't have them. – Camilo Martin Nov 24, 2010 at 19:47

6 GTK - 'scuse me, GObject - is actually a pretty good example of OOP (sorta) in C. So, to answer @Camilo, for C interpoliability. – new123456 Dec 14, 2010 at 22:59

6 It's really shocking how this question could have been closed. It's a very good question, an eye opener, as the received points tell. It just tells how SO is driven... – cesss Apr 12, 2020 at 19:07

28 Answers Sorted by: Highest score (default)

392

Since you're talking about polymorphism then yes, you can, we were doing that sort of stuff years before C++ came about.

Basically you use a `struct` to hold both the data and a list of function pointers to point to the relevant functions for that data.

So, in a communications class, you would have an open, read, write and close call which would be maintained as four function pointers in the structure, alongside the data for an object, something like:

```
typedef struct {
    int (*open)(void *self, char *fspec);
    int (*close)(void *self);
    int (*read)(void *self, void *buff, size_t max_sz,
    int (*write)(void *self, void *buff, size_t max_sz
    // And data goes here.
} tCommClass;

tCommClass commRs232;
commRs232.open = &rs232Open;
: :
commRs232.write = &rs232Write;

tCommClass commTcp;
commTcp.open = &tcpOpen;
: :
commTcp.write = &tcpWrite;
```

Of course, those code segments above would actually be in a "constructor" such as `rs232Init()`.

When you 'inherit' from that class, you just change the pointers to point to your own functions. Everyone that called those functions would do it through the function pointers, giving you your polymorphism:

```
int stat = (commTcp.open)(commTcp, "bigiron.box.com:50
```

Sort of like a manual vtable.

You could even have virtual classes by setting the pointers to NULL -the behaviour would be slightly different to C++ (a core dump at run-time rather than an error at compile time).

Here's a piece of sample code that demonstrates it. First the top-level class structure:

```
#include <stdio.h>

// The top-level class.

typedef struct sCommClass {
    int (*open)(struct sCommClass *self, char *fspec);
} tCommClass;
```

Then we have the functions for the TCP 'subclass':

```
// Function for the TCP 'class'.

static int tcpOpen (tCommClass *tcp, char *fspec) {
    printf ("Opening TCP: %s\n", fspec);
```

```
    return 0;
}
static int tcpInit (tCommClass *tcp) {
    tcp->open = &tcpOpen;
    return 0;
}
```

And the HTTP one as well:

```
// Function for the HTTP 'class'.

static int httpOpen (tCommClass *http, char *fspec) {
    printf ("Opening HTTP: %s\n", fspec);
    return 0;
}
static int httpInit (tCommClass *http) {
    http->open = &httpOpen;
    return 0;
}
```

And finally a test program to show it in action:

```
// Test program.

int main (void) {
    int status;
    tCommClass commTcp, commHttp;

    // Same 'base' class but initialised to different

    tcpInit (&commTcp);
    httpInit (&commHttp);

    // Called in exactly the same manner.

    status = (commTcp.open)(&commTcp, "bigiron.box.com
    status = (commHttp.open)(&commHttp, "http://www.mi
```

```
    return 0;
}
```

This produces the output:

```
Opening TCP: bigiron.box.com:5000
Opening HTTP: http://www.microsoft.com
```

so you can see that the different functions are being called, depending on the sub-class.

Share Improve this answer Follow

edited May 12, 2017 at 0:39

community wiki
6 revs, 2 users 92%
paxdiablo

62 Encapsulation is pretty easy, polymorphism is doable - but inheritence is tricky – Martin Beckett Jul 19, 2010 at 16:10

6 lwn.net recently published an article titled Object Oriented design Patterns in the kernel on the subject of stucts similar to the above answer - that is, a struct containing function pointers, or a pointer to a struct that has functions that take a pointer to the struct with the data we are working with as a parameter. – user107498 Jun 5, 2011 at 0:51

15 +1 Nice example! Although if anyone really wants to go down this road, it would be more appropriate for "instance" structs to have a *single field* pointing to their "virtual table" instance, containing all the virtual functions for that type at one place. I.e. your `tCommClass` would be renamed into `tCommVT`, and a `tCommClass` struct would only have data fields and a single `tCommVT vt` field pointing to the "one and only"

virtual-table. Carrying all pointers around with each instance adds unnecessary overhead and resembles more of how you would do stuff in JavaScript than C++, IMHO. – vgru Oct 23, 2013 at 7:45

1 So this demonstrates the implementation of a single interface, but whatabout implementing multiple interfaces? Or multiple inheritance? – weberc2 Jul 7, 2014 at 15:35

3 Weber, if you want all the functionality of C++, you probably should be using C++. The question asked specifically about polymorphism, the ability of objects to take a different "form". You can certainly do interfaces and multiple inheritence in C but it's a fair bit of extra work, and you have to manage the smarts yourself rather than using C++ built-in stuff. – paxdiablo Jul 7, 2014 at 22:05

102

Namespaces are often done by doing:

```
stack_push(thing *)
```

instead of

```
stack::push(thing *)
```

To make a C struct into something like a C++ class you can turn:

```
class stack {
     public:
        stack();
        void push(thing *);
        thing * pop();
        static int this_is_here_as_an_example_only;
```

```
    private:
        ...
};
```

Into

```
struct stack {
    struct stack_type * my_type;
    // Put the stuff that you put after private: here
};
struct stack_type {
    void (* construct)(struct stack * this); // This
memory
    struct stack * (* operator_new)(); // This alloca
it to construct, and then returns it
    void (*push)(struct stack * this, thing * t); //
    thing * (*pop)(struct stack * this); // Pops the
and returns it
    int this_is_here_as_an_example_only;
}Stack = {
    .construct = stack_construct,
    .operator_new = stack_operator_new,
    .push = stack_push,
    .pop = stack_pop
};
// All of these functions are assumed to be defined so
```

And do:

```
struct stack * st = Stack.operator_new(); // Make a ne
if (!st) {
   // Do something about it
} else {
   // You can use the stack
   stack_push(st, thing0); // This is a non-virtual ca
   Stack.push(st, thing1); // This is like casting *st
already is) and doing the push
   st->my_type.push(st, thing2); // This is a virtual
}
```

I didn't do the destructor or delete, but it follows the same pattern.

this_is_here_as_an_example_only is like a static class variable -- shared among all instances of a type. All methods are really static, except that some take a this *

Share Improve this answer Follow

edited Dec 31, 2015 at 16:14

community wiki
5 revs, 2 users 88%
nategoose

2 @nategoose - `st->my_type->push(st, thing2);` instead of `st->my_type.push(st, thing2);` – Fabricio Jun 3, 2012 at 12:48

@nategoose: **OR** `struct stack_type my_type;` instead of `struct stack_type * my_type;` – Fabricio Jun 3, 2012 at 16:18

3 I like the concept of having a struct for the class. But how about a generic `Class` struct? That would make the OO C *more* dynamic than C++. How about that? By the way, +1. – Linuxios Jul 28, 2012 at 20:21

67

I believe that besides being useful in its own right, implementing OOP in C is an excellent way to **learn** OOP and understand its inner workings. Experience of many programmers has shown that to use a technique efficiently and confidently, a programmer must understand how the underlying concepts are ultimately implemented. Emulating classes, inheritance, and polymorphism in C teaches just this.

To answer the original question, here are a couple resources that teach how to do OOP in C:

EmbeddedGurus.com blog post "Object-based programming in C" shows how to implement classes and single inheritance in portable C: http://embeddedgurus.com/state-space/2008/01/object-based-programming-in-c/

Application Note ""C+"—Object Oriented Programming in C" shows how to implement classes, single inheritance, and late binding (polymorphism) in C using preprocessor macros: http://www.state-machine.com/resources/cplus_3.0_manual.pdf, the example code is available from http://www.state-machine.com/resources/cplus_3.0.zip

edited Apr 29, 2010 at 1:37

community wiki

2 revs
Miro

6 New url for the C+ manual: state-machine.com/doc/cplus_3.0_manual.pdf – Liang Jan 13, 2016 at 14:27

32

I've seen it done. I wouldn't recommend it. C++ originally started this way as a preprocessor that produced C code as an intermediate step.

Essentially what you end up doing is create a dispatch table for all of your methods where you store your function references. Deriving a class would entail copying this dispatch table and replacing the entries that you wanted to override, with your new "methods" having to call the original method if it wants to invoke the base method. Eventually, you end up rewriting C++.

Share Improve this answer Follow

answered Dec 9, 2008 at 4:04

community wiki
tvanfosson

8 "Eventually, you end up rewriting C++" I wondered if/feared that would be the case. – Dinah Apr 14, 2010 at 15:53

47 Or, you might end up rewriting Objective C, which would be a much more attractive outcome. – Prof. Falken Nov 5, 2010 at 12:15

3 There is the class-less flavour of OOP, such as in Javascript, where the guru says: "We don't need classes to make lots of similar objects." But I fear this is not easy to achieve in C. Not (yet) in a position to tell, though. (Is there a clone() routine to clone a struct?) – Lumi Jun 11, 2011 at 21:32

1 Another smart guys, who had to actually *implement* that and make that implementation *fast* (Google, V8 engine) have done everything do add (hidden) classes to JavaScript back. – cubuspl42 Feb 10, 2013 at 21:03

Isn't `glib` written in C in objective way? – kravemir Feb 6, 2018 at 7:42

27

Sure that is possible. This is what GObject, the framework that all of GTK+ and GNOME is based on, does.

Share Improve this answer Follow

edited Dec 31, 2015 at 15:57

community wiki
3 revs, 2 users 53%
Peter Mortensen

What are pros/cons of such approach? Ie. it's much easier to write it using C++. – kravemir Feb 6, 2018 at 7:42

@kravemir Well, C++ is not quite as portable as C, and it's a bit harder to link C++ to code that might be compiled by a different C++ compiler. But yes, it is easier to write classes in C++, although GObject isn't really that difficult either (assuming you don't mind a little boiler plate). – Edwin Buck Apr 18, 2018 at 4:46

I dislike how `GObject` forces you to use a constructor which invariably allocates objects on the stack. I prefer out-of-line constructors that allow you to decide how the memory is allocated, on the stack, heap, static, etc. I **want** to **have** to call `free()` when I am done, or not if I am not using the heap. – CPlus May 13 at 1:52

18

The C stdio FILE sub-library is an excellent example of how to create abstraction, encapsulation, and modularity in unadulterated C.

Inheritance and polymorphism - the other aspects often considered essential to OOP - do not necessarily provide the productivity gains they promise and reasonable arguments have been made that they can actually hinder development and thinking about the problem domain.

Share Improve this answer Follow

edited Oct 24, 2016 at 11:58

community wiki
2 revs, 2 users 93%
msw

Isn't stdio abstracted on kernel layer? If I'm not mistaken, C-library treats them as character files/devices, and kernel drivers do the job,... – kravemir Feb 6, 2018 at 7:44

Trivial example with an Animal and Dog: You mirror C++'s vtable mechanism (largely anyway). You also separate

16

allocation and instantiation (Animal_Alloc, Animal_New) so we don't call malloc() multiple times. We must also explicitly pass the `this` pointer around.

If you were to do non-virtual functions, that's trival. You just don't add them to the vtable and static functions don't require a `this` pointer. Multiple inheritance generally requires multiple vtables to resolve ambiguities.

Also, you should be able to use setjmp/longjmp to do exception handling.

```
struct Animal_Vtable{
    typedef void (*Walk_Fun)(struct Animal *a_This);
    typedef struct Animal * (*Dtor_Fun)(struct Animal

    Walk_Fun Walk;
    Dtor_Fun Dtor;
};

struct Animal{
    Animal_Vtable vtable;

    char *Name;
};

struct Dog{
    Animal_Vtable vtable;

    char *Name; // Mirror member variables for easy ac
    char *Type;
};

void Animal_Walk(struct Animal *a_This){
    printf("Animal (%s) walking\n", a_This->Name);
}

struct Animal* Animal_Dtor(struct Animal *a_This){
    printf("animal::dtor\n");
```

```
    return a_This;
}

Animal *Animal_Alloc(){
    return (Animal*)malloc(sizeof(Animal));
}

Animal *Animal_New(Animal *a_Animal){
    a_Animal->vtable.Walk = Animal_Walk;
    a_Animal->vtable.Dtor = Animal_Dtor;
    a_Animal->Name = "Anonymous";
    return a_Animal;
}

void Animal_Free(Animal *a_This){
    a_This->vtable.Dtor(a_This);

    free(a_This);
}

void Dog_Walk(struct Dog *a_This){
    printf("Dog walking %s (%s)\n", a_This->Type, a_Th
}

Dog* Dog_Dtor(struct Dog *a_This){
    // Explicit call to parent destructor
    Animal_Dtor((Animal*)a_This);

    printf("dog::dtor\n");

    return a_This;
}

Dog *Dog_Alloc(){
    return (Dog*)malloc(sizeof(Dog));
}

Dog *Dog_New(Dog *a_Dog){
    // Explict call to parent constructor
    Animal_New((Animal*)a_Dog);

    a_Dog->Type = "Dog type";
    a_Dog->vtable.Walk = (Animal_Vtable::Walk_Fun) Dog
    a_Dog->vtable.Dtor = (Animal_Vtable::Dtor_Fun) Dog
```

```
    return a_Dog;
}

int main(int argc, char **argv){
    /*
      Base class:

        Animal *a_Animal = Animal_New(Animal_Alloc());
    */
    Animal *a_Animal = (Animal*)Dog_New(Dog_Alloc());

    a_Animal->vtable.Walk(a_Animal);

    Animal_Free(a_Animal);
}
```

PS. This is tested on a C++ compiler, but it should be easy to make it work on a C compiler.

Share Improve this answer Follow

edited Dec 31, 2015 at 16:02

community wiki
2 revs, 2 users 73%
Jasper Bekkers

`typedef` inside a `struct` is not possible in C. – masoud Jul 24, 2019 at 13:14

14

There are several techniques that can be used. The most important one is more how to split the project. We use an interface in our project that is declared in a .h file and the implementation of the object in a .c file. The important



part is that all modules that include the .h file see only an object as a `void *`, and the .c file is the only module who knows the internals of the structure.

Something like this for a class we name FOO as an example:

In the .h file

```
#ifndef FOO_H_
#define FOO_H_

...
 typedef struct FOO_type FOO_type;     /* That's all t
knows about FOO */

/* Declaration of accessors, functions */
FOO_type *FOO_new(void);
void FOO_free(FOO_type *this);
...
void FOO_dosomething(FOO_type *this, param ...):
char *FOO_getName(FOO_type *this, etc);
#endif
```

The C implementation file will be something like that.

```
#include <stdlib.h>
...
#include "FOO.h"

struct FOO_type {
    whatever...
};


FOO_type *FOO_new(void)
{
    FOO_type *this = calloc(1, sizeof (FOO_type));
```

```
    ...
    FOO_dosomething(this, );
    return this;
}
```

So I give the pointer explicitly to an object to every function of that module. A C++ compiler does it implicitly, and in C we write it explicitly out.

I really use `this` in my programs, to make sure that my program does not compile in C++, and it has the fine property of being in another color in my syntax highlighting editor.

The fields of the FOO_struct can be modified in one module and another module doesn't even need to be recompiled to be still usable.

With that style I already handle a big part of the advantages of OOP (data encapsulation). By using function pointers, it's even easy to implement something like inheritance, but honestly, it's really only rarely useful.

edited Dec 31, 2015 at 16:12

community wiki
4 revs, 2 users 83%
Patrick Schlüter

7 If you do `typedef struct FOO_type FOO_type` instead of a typedef to void in the header you get the added benefit of

type checking, while still not exposing your structure.
– Scott Wales Apr 29, 2010 at 0:38

14

This has been interesting to read. I have been pondering the same question myself, and the benefits of thinking about it are this:

- Trying to imagine how to implement OOP concepts in a non-OOP language helps me understand the strengths of the OOp language (in my case, C++). This helps give me better judgement about whether to use C or C++ for a given type of application -- where the benefits of one out-weighs the other.
- In my browsing the web for information and opinions on this I found an author who was writing code for an embedded processor and only had a C compiler available: http://www.eetimes.com/discussion/other/4024626/Object-Oriented-C-Creating-Foundation-Classes-Part-1

In his case, analyzing and adapting OOP concepts in plain C was a valid pursuit. It appears he was open to sacrificing some OOP concepts due to the performance overhead hit resulting from attempting to implement them in C.

The lesson I've taken is, yes it can be done to a certain degree, and yes, there are some good reasons to attempt it.

In the end, the machine is twiddling stack pointer bits, making the program counter jump around and calculating memory access operations. From the efficiency standpoint, the fewer of these calculations done by your program, the better... but sometimes we have to pay this tax simply so we can organize our program in a way that makes it least susceptible to human error. The OOP language compiler strives to optimize both aspects. The programmer has to be much more careful implementing these concepts in a language like C.

Share Improve this answer Follow

edited Dec 31, 2015 at 16:28

community wiki
3 revs, 3 users 69%
RJB

13

Check out GObject. It's meant to be OO in C and one implementation of what you're looking for. If you really want OO though, go with C++ or some other OOP language. GObject can be really tough to work with at times if you're used to dealing with OO languages, but like anything, you'll get used to the conventions and flow.

Share Improve this answer Follow

answered Apr 28, 2010 at 19:53

community wiki
NG.

10

You may find it helpful to look at Apple's documentation for its Core Foundation set of APIs. It is a pure C API, but many of the types are bridged to Objective-C object equivalents.

You may also find it helpful to look at the design of Objective-C itself. It's a bit different from C++ in that the object system is defined in terms of C functions, e.g. `objc_msg_send` to call a method on an object. The compiler translates the square bracket syntax into those function calls, so you don't have to know it, but considering your question you may find it useful to learn how it works under the hood.

Share Improve this answer Follow

answered Apr 28, 2010 at 19:58

community wiki
benzado

8

You can fake it using function pointers, and in fact, I think it is theoretically possible to compile C++ programs into C.

However, it rarely makes sense to force a paradigm on a language rather than to pick a language that uses a paradigm.

Share Improve this answer Follow

answered Dec 9, 2008 at 4:05

community wiki
Uri

5 The very first C++ compiler did exactly that - it converted the C++ code into equivalent (but ugly and non-human-readable) C code, which was then compiled by the C compiler. – Adam Rosenfield Dec 9, 2008 at 6:05

2 EDG, Cfront and some others are still capable of doing this. With a very good reason: not every platform has a C++ compiler. – Jasper Bekkers Dec 9, 2008 at 9:50

For some reason I thought that C-front only supported certain C++ extensions (e.g., references) but not full OOP / dynamic dispatch emulation. – Uri Dec 9, 2008 at 20:15

2 You can also do the same thing with LLVM and the C backend. – Zifre Mar 31, 2009 at 17:47

8

Yes, you can. People were writing object-oriented C before C++ or Objective-C came on the scene. Both C++ and Objective-C were, in parts, attempts to take some of the OO concepts used in C and formalize them as part of the language.

Here's a really simple program that shows how you can make something that looks-like/is a method call (there are better ways to do this. This is just proof the language supports the concepts):

```c
#include<stdio.h>

struct foobarbaz{
    int one;
    int two;
    int three;
    int (*exampleMethod)(int, int);
};

int addTwoNumbers(int a, int b){
    return a+b;
}

int main()
{
    // Define the function pointer
    int (*pointerToFunction)(int, int) = addTwoNumbers

    // Let's make sure we can call the pointer
    int test = (*pointerToFunction)(12,12);
    printf ("test: %u \n",  test);

    // Now, define an instance of our struct
    // and add some default values.
    struct foobarbaz fbb;
    fbb.one   = 1;
    fbb.two   = 2;
    fbb.three = 3;

    // Now add a "method"
    fbb.exampleMethod = addTwoNumbers;

    // Try calling the method
    int test2 = fbb.exampleMethod(13,36);
    printf ("test2: %u \n",  test2);

    printf("\nDone\n");
```

```
    return 0;
}
```

Share Improve this answer Follow

edited Dec 31, 2015 at 16:00

community wiki
2 revs, 2 users 80%
Alan Storm

7

Object oriented C, can be done, I've seen that type of code in production in Korea, and it was the most horrible monster I'd seen in years (this was like last year(2007) that I saw the code). So yes it can be done, and yes people have done it before, and still do it even in this day and age. But I'd recommend C++ or Objective-C, both are languages born from C, with the purpose of providing object orientation with different paradigms.

Share Improve this answer Follow

answered Dec 9, 2008 at 4:09

community wiki
Robert Gould

6

If you are convinced that an OOP approach is superior for the problem you are trying to solve, why would you be trying to solve it with a non-OOP language? It seems like

you're using the wrong tool for the job. Use C++ or some other object-oriented C variant language.

If you are asking because you are starting to code on an already existing large project written in C, then you shouldn't try to force your own (or anyone else's) OOP paradigms into the project's infrastructure. Follow the guidelines that are already present in the project. In general, clean APIs and isolated libraries and modules will go a long way towards having a clean OOP-**ish** design.

If, after all this, you really are set on doing OOP C, read this (PDF).

Share Improve this answer Follow

edited Apr 28, 2010 at 19:49

community wiki
2 revs
RarrRarrRarr

41 Not really answering the question... – Brian Postow Apr 28, 2010 at 21:29

2 @Brian, the link to the PDF would appear to answer the question directly, although I haven't had time to check for myself. – Mark Ransom Apr 28, 2010 at 21:40

5 The link to the PDF appears to be an entire textbook on the subject... A beautiful proof, but it doesn't fit into the margin... – Brian Postow Apr 28, 2010 at 22:46

5 yes, answer the question. it's perfectly valid to ask how to use a language in a particular way. there was no request for opinions on other languages.... – Tim Ring Apr 28, 2010 at 23:50

9 @Brian & Tim Ring: The question asked for *book recommendations* on a topic; I gave him a link to a *book* that specifically addresses this topic. I also gave my opinion on why the approach to the problem may not be optimal (which I think many people on here seem to agree with, based on votes and other comments/answers). Do you have any suggestions for improving my answer? – RarrRarrRarr Apr 29, 2010 at 0:16

6

A little OOC code to add:

```
#include <stdio.h>

struct Node {
    int somevar;
};

void print() {
    printf("Hello from an object-oriented C method!");
};

struct Tree {
    struct Node * NIL;
    void (*FPprint)(void);
    struct Node *root;
    struct Node NIL_t;
} TreeA = {&TreeA.NIL_t,print};

int main()
{
    struct Tree TreeB;
    TreeB = TreeA;
    TreeB.FPprint();
```

```
    return 0;
}
```

Share Improve this answer Follow

edited Dec 31, 2015 at 16:29

community wiki
2 revs, 2 users 77%
user922475

5

I've been digging this for one year:

As the GObject system is hard to use with pure C, I tried to write some nice macros to ease the OO style with C.

```
#include "OOStd.h"

CLASS(Animal) {
    char *name;
    STATIC(Animal);
    vFn talk;
};
static int Animal_load(Animal *THIS,void *name) {
    THIS->name = name;
    return 0;
}
ASM(Animal, Animal_load, NULL, NULL, NULL)

CLASS_EX(Cat,Animal) {
    STATIC_EX(Cat, Animal);
};
static void Meow(Animal *THIS){
    printf("Meow!My name is %s!\n", THIS->name);
}

static int Cat_loadSt(StAnimal *THIS, void *PARAM){
    THIS->talk = (void *)Meow;
```

```
    return 0;
}
ASM_EX(Cat,Animal, NULL, NULL, Cat_loadSt, NULL)


CLASS_EX(Dog,Animal){
    STATIC_EX(Dog, Animal);
};

static void Woof(Animal *THIS){
    printf("Woof!My name is %s!\n", THIS->name);
}

static int Dog_loadSt(StAnimal *THIS, void *PARAM) {
    THIS->talk = (void *)Woof;
    return 0;
}
ASM_EX(Dog, Animal, NULL, NULL, Dog_loadSt, NULL)

int main(){
    Animal *animals[4000];
    StAnimal *f;
    int i = 0;
    for (i=0; i<4000; i++)
    {
        if(i%2==0)
            animals[i] = NEW(Dog,"Jack");
        else
            animals[i] = NEW(Cat,"Lily");
    };
    f = ST(animals[0]);
    for(i=0; i<4000; ++i) {
        f->talk(animals[i]);
    }
    for (i=0; i<4000; ++i) {
        DELETE0(animals[i]);
    }
    return 0;
}
```

Here is my project site (I don't have enough time to write en. doc,however the doc in chinese is much better).

OOC-GCC

Share Improve this answer Follow

edited Dec 31, 2015 at 16:34

community wiki
3 revs, 2 users 73%
dameng

the **CLASS STATIC ASM NEW DELETE ST ...** are macros in the OOC-GCC – dameng Dec 5, 2011 at 12:06

4

One thing you might want to do is look into the implementation of the Xt toolkit for X Window. Sure it is getting long in the tooth, but many of the structures used were designed to work in an OO fashion within traditional C. Generally this means adding an extra layer of indirection here and there and designing structures to lay over each other.

You can really do lots in the way of OO situated in C this way, even though it feels like it some times, OO concepts did not spring fully formed from the mind of `#include<favorite_OO_Guru.h>`. They really constituted many of the established best practice of the time. OO languages and systems only distilled and amplified parts of the programing zeitgeist of the day.

Share Improve this answer Follow

edited Dec 31, 2015 at 16:26

community wiki
4 revs, 3 users 50%
Peter Mortensen

4

The answer to the question is 'Yes, you can'.

Object-oriented C (OOC) kit is for those who want to program in an object-oriented manner, but sticks on the good old C as well. OOC implements classes, single and multiple inheritance, exception handling.

Features

• Uses only C macros and functions, no language extensions required! (ANSI-C)

• Easy-to-read source code for your application. Care was taken to make things as simple as possible.

• Single inheritance of classes

• Multiple inheritance by interfaces and mixins (since version 1.3)

• Implementing exceptions (in pure C!)

• Virtual functions for classes

• External tool for easy class implementation

For more details, visit http://ooc-coding.sourceforge.net/.

Share Improve this answer Follow

edited Dec 31, 2015 at 16:36

community wiki
2 revs, 2 users 77%
Sachin Mhetre

4

It's seem like people are trying emulate the C++ style using C. My take is that doing object-oriented programming C is really doing struct-oriented programming. However, you can achieve things like late binding, encapsulation, and inheritance. For inheritance you explicitly define a pointer to the base structs in your sub struct and this is obviously a form of multiple inheritance. You'll also need to determine if your

```
//private_class.h
struct private_class;
extern struct private_class * new_private_class();
extern int ret_a_value(struct private_class *, int a,
extern void delete_private_class(struct private_class
void (*late_bind_function)(struct private_class *p);

//private_class.c
struct inherited_class_1;
struct inherited_class_2;

struct private_class {
  int a;
  int b;
  struct inherited_class_1 *p1;
  struct inherited_class_2 *p2;
};

struct inherited_class_1 * new_inherited_class_1();
struct inherited_class_2 * new_inherited_class_2();
```

```
struct private_class * new_private_class() {
  struct private_class *p;
  p = (struct private_class*) malloc(sizeof(struct pri
  p->a = 0;
  p->b = 0;
  p->p1 = new_inherited_class_1();
  p->p2 = new_inherited_class_2();
  return p;
}

    int ret_a_value(struct private_class *p, int a, in
      return p->a + p->b + a + b;
    }

    void delete_private_class(struct private_class *p)
      //release any resources
      //call delete methods for inherited classes
      free(p);
    }
    //main.c
    struct private_class *p;
    p = new_private_class();
    late_bind_function = &implementation_function;
    delete_private_class(p);
```

compile with `c_compiler main.c inherited_class_1.obj inherited_class_2.obj  private_class.obj`.

So the advice is to stick to a pure C style and not try to force into a C++ style. Also this way lends itself to a very clean way of building an API.

Share Improve this answer edited Dec 31, 2015 at 16:38

Follow

community wiki

4 revs, 2 users 91%
user2074102

For inheritance typically the base class or instance structure is embedded in the derived one, not allocated separately and referred using pointers. That way the topmost base is always at the start of any of its derived types' structures, so they can be cast to each other with ease, which you can't do with pointers that might be at any offset. – underscore_d May 11, 2017 at 13:06

3

OOP is only a paradigm which place datas as more important than code in programs. OOP is not a language. So, like plain C is a simple language, OOP in plain C is simple too.

Share Improve this answer Follow

answered Jan 15, 2012 at 23:50

community wiki
anonyme

2

See http://slkpg.byethost7.com/instance.html for yet another twist on OOP in C. It emphasizes instance data for reentrancy using just native C. Multiple inheritance is done manually using function wrappers. Type safety is maintained. Here is a small sample:

```
typedef struct _peeker
{
    log_t     *log;
```

```
    symbols_t *sym;
    scanner_t  scan;            // inherited instance
    peek_t     pk;
    int        trace;

    void    (*push) ( SELF *d, symbol_t *symbol );
    short   (*peek) ( SELF *d, int level );
    short   (*get)  ( SELF *d );
    int     (*get_line_number) ( SELF *d );

} peeker_t, SlkToken;

#define push(self,a)            (*self).push(self, a)
#define peek(self,a)            (*self).peek(self, a)
#define get(self)               (*self).get(self)
#define get_line_number(self)   (*self).get_line_numbe

INSTANCE_METHOD
int
(get_line_number) ( peeker_t *d )
{
    return  d->scan.line_number;
}

PUBLIC
void
InitializePeeker ( peeker_t  *peeker,
                   int        trace,
                   symbols_t *symbols,
                   log_t     *log,
                   list_t    *list )
{
    InitializeScanner ( &peeker->scan, trace, symbols,
    peeker->log = log;
    peeker->sym = symbols;
    peeker->pk.current = peeker->pk.buffer;
    peeker->pk.count = 0;
    peeker->trace = trace;

    peeker->get_line_number = get_line_number;
    peeker->push = push;
    peeker->get = get;
```

```
    peeker->peek = peek;
}
```

Share Improve this answer Follow

edited Jul 28, 2012 at 13:08

community wiki
3 revs
slkpg

2

I'm a bit late to the party, but I want to share my experience on the topic: I work with embedded stuff these days, and the only (reliable) compiler I have is C, so that I want to apply object-oriented approach in my embedded projects written in C.

Most of the solutions I've seen so far use typecasts heavily, so we lose type safety: compiler won't help you if you make a mistake. This is completely unacceptable.

Requirements that I have:

- Avoid typecasts as much as possible, so we don't lose type safety;
- Polymorphism: we should be able to use virtual methods, and user of the class should not be aware whether some particular method is virtual or not;
- Multiple inheritance: I don't use it often, but sometimes I really want some class to implement

multiple interfaces (or to extend multiple superclasses).

I've explained my approach in detail in this article: Object-oriented programming in C; plus, there is an utility for autogeneration of boilerplate code for base and derived classes.

Share Improve this answer Follow

edited Mar 19, 2015 at 13:57

community wiki
2 revs
Dmitry Frank

2

I built a little library where I tried that and to me it works real nicely. So I thought I share the experience.

https://github.com/thomasfuhringer/oxygen

Single inheritance can be implemented quite easily using a struct and extending it for every other child class. A simple cast to the parent structure makes it possible to use parent methods on all the descendants. As long as you know that a variable points to a struct holding this kind of an object you can always cast to the root class and do introspection.

As has been mentioned, virtual methods are somewhat trickier. But they are doable. To keep things simple I just use an array of functions in the class description structure

which every child class copies and repopulates individual slots where required.

Multiple inheritance would be rather complicated to implement and comes with a significant performance impact. So I leave it. I do consider it desirable and useful in quite a few cases to cleanly model real life circumstances, but in probably 90% of cases single inheritance covers the needs. And single inheritance is simple and costs nothing.

Also I do not care about type safety. I think you should not depend on the compiler to prevent you from programming mistakes. And it shields you only from a rather small part of errors anyway.

Typically, in an object oriented environment you also want to implement reference counting to automate memory management to the extent possible. So I also put a reference count into the “Object” root class and some functionality to encapsulate allocation and deallocation of heap memory.

It is all very simple and lean and gives me the essentials of OO without forcing me to deal with the monster that is C++. And I retain the flexibility of staying in C land, which among other things makes it easier to integrate third party libraries.

answered Mar 2, 2016 at 8:38

community wiki
Thomas F.

1

Yes, it is possible.

This is pure C, no macros preprocessing. It has inheritance, polymorphism, data encapsulation (including private data). It does not have equivalent protected qualifier, wich means private data is private down the inheritance chain too.

```
#include "triangle.h"
#include "rectangle.h"
#include "polygon.h"

#include <stdio.h>

int main()
{
    Triangle tr1= CTriangle->new();
    Rectangle rc1= CRectangle->new();

    tr1->width= rc1->width= 3.2;
    tr1->height= rc1->height= 4.1;

    CPolygon->printArea((Polygon)tr1);

    printf("\n");

    CPolygon->printArea((Polygon)rc1);
}

/*output:
6.56
13.12
*/
```

Share Improve this answer

Follow

edited Oct 26, 2016 at 19:47

Félix Adriyel Gagnon-Grenier

8,874 ● 10 ● 56 ● 67

answered Jul 24, 2012 at 10:05

rogergc

29 ● 1

Here is the pastebin where one can see more of this code: pastebin.com/bLxTP8tA – josch Sep 18, 2014 at 4:30

Yes, but I have never seen anyone attempt to implement any sort of polymorphism with C.

0

Share Improve this answer

Follow

answered Dec 9, 2008 at 4:06

community wiki

Paul Morel

6 You need to look around more :) For instance, Microsoft's Direct X has a polymorphic C interface. – AShelly Dec 9, 2008 at 19:09

10 Look into linux kernel implementation for example. It is very common and widely used practice in C. – Ilya Dec 12, 2008 at 3:02

3 also glib is polymorphic, or can be used in a way that allows polymorphism (it's like C++ you have to explicitly say which calls are virtual) – Spudd86 Aug 31, 2010 at 18:44

1 Polymorphism is not that rare in C, on the other hand multiple inheritance is. – Johan Bjäreholt Jan 25, 2018 at 15:26

0

I think that the first thing to say is that (IMHO at least) C's implementation of function pointers is REALLY hard to use. I would jump through a WHOLE lot of hoops to avoid function pointers...

that said, I think that what other people have said is pretty good. you have structures, you have modules, instead of `foo->method(a,b,c)`, you end up with `method(foo,a,b,c)` If you have more than one type with a "method" method, then you can prefix it with the type, so `FOO_method(foo,a,b,c)`, as others have said... with good use of .h files you can get private and public, etc.

Now, there are a few things that this technique WON'T give you. It won't give you private data fields. that, I think, you have to do with willpower and good coding hygiene... Also, there isn't an easy way to do inheritance with this.

Those are the easy parts at least...the rest, I think is a 90/10 kind of situation. 10% of the benefit will require 90% of the work...

Share Improve this answer Follow

answered Apr 28, 2010 at 21:37

community wiki

Brian Postow

1 Single inheritance (without polymorphism, though) can be quite easily implemented with this technique as well. All you need to to is embed the superclass as the first member of the subclass. By the C Standard, the whole structure must necessarily be aligned with the first member, so any method designed for foo (method(foo, a, b, c)) will work when a bar pointer is passed instead (bar being a subclass of foo). This is inheritance. – Miro Samek Apr 29, 2010 at 0:31

@miro. Wow. that's ... that's a serious kludge right there... – Brian Postow Apr 29, 2010 at 13:29

1 not really... it's used all over the place... glib is pretty much built on this idea, Linux kernel (extensively), also it's essentially the same thing that happens when you use an object oriented language, the compiler uses the same object layout that is described above (multiple inheritance complicates this slightly though, one of the superclasses must be at an offset from the object start) – Spudd86 Aug 31, 2010 at 18:49

1 you can get private data by passing pointers to things that are declared, but not defined anywhere outside the implementation specifics. So the `foo` parameter is a `struct foo *` that nobody has to know what's really in it. – Flexo - Save the data dump ♦ Aug 23, 2011 at 20:45

@MiroSamek Sure, it'll work, but the pointers must explicitly be cast to the type that the function expects. C doesn't formally recognise inheritance relationships and so will not implicitly cast pointers/references like C++ or other languages would. – underscore_d May 11, 2017 at 13:10

I propose to use Objective-C, which is a superset of C.

0

While Objective-C is 30 years old, it allows to write elegant code.

http://en.wikipedia.org/wiki/Objective-C

Share Improve this answer Follow

answered Aug 27, 2010 at 22:42

community wiki
SteAp

1 In that case I would recommend C++ instead since its actually object oriented... – yyny Oct 1, 2015 at 12:42

This is not an answer. But anyway, @YoYoYonnY: I don't use Objective-C and do use C++, but comments like that are of no use without basis, and you've provided none. Why do you claim Objective-C falls short of being "actually object oriented..."? And why does C++ succeed where Objective-C fails? The funny thing is that Objective-C, well, literally has the word *Object* in its name, whereas C++ markets itself as a multi-paradigm language, not an OOP one (i.e. not primarily OOP, & in some rather extreme folk's view not OOP at all)... so are you sure you didn't get those names the wrong way round? – underscore_d May 12, 2017 at 17:07