

Forum: [Hack Forums](#)

Thread: [way my GOLANG boys at?](#)

Fugg - June 21, 2022, 2:48 pm

I just recently started playing around with GoLang. Wow. I love it. Especially with CGo... is really awesome. Programming in Go is very straight forward and easy, and most syscalls and whatnot are already in official packages. Anything else can be done with C using CGo. I haven't been doing hardly any systems programming or anything like that (mostly backend javascript web stuff for me)... and GO has been a GREAT language to get into some system functions and shit.

I haven't used CGo at all really yet. Last time I coded C/C++ was ~2013... so, needless to say... I forgot everything that I know about C.

I am having a great time using Golang to research how RunPE works and developing my own flavor.

Any Gophers out here?

Faith Connors - June 21, 2022, 3:39 pm

for a systems language it seems to suck a lot of ass at programming systems lol

i tried it (and tried to shill it) around 2018 when it was at the peak of its hype (even one of my university's classes used it to teach paradigms) and honestly i just kept finding flaws over and over with it.

--> no pointer-arithmetic or anything considered 'unsafe'

--> helloworld binary size at 2mb, managed to get it down to 600kb by stripping symbol table and using write() syscall over fmt (no linking required), still 590kb bigger than C equivalent

--> too much syntactic sugar to do the same task in 3 different ways, making reading someone else's code borderline impossible

and... a few gripes with the community-aspect of it...

--> community always too busy just writing wrappers for any useful library, rather than actually writing a useful library of their own. then abandoning the project after the 1.0 release, leaving the wrapper to be obsolete after like a year.

--> can't decide on toolchain design, see [this thread](#). devs make borderline subhuman iq decision, breaks the last 6 years or so of projects as a result, then tell the community that the community is just too stupid to understand how smart and exciting their "changes" are.

in short: imo language has almost no future.

the community acts like children when they get a new toy, then get bored with all their older toys when they see the big new shiny thing.

tech-wise, it does some of the things other languages already do (c, cpp, rust, ada,) but seems to do every single one of those few things worse.

maybe if they decide to develop it solely as a web-backend language and ditch any promises of

actual systems development, they might have a future.

Fugg - June 21, 2022, 3:49 pm

(Jun 21, 2022 03:39 PM)Faith Connors Wrote: [\[CITE\]](#)

for a systems language it seems to suck a lot of ass at programming systems lol

i tried it (and tried to shill it) around 2018 when it was at the peak of its hype (even one of my university's classes used it to teach paradigms) and honestly i just kept finding flaws over and over with it.

--> no pointer-arithmetic or anything considered 'unsafe'

--> helloworld binary size at 2mb, managed to get it down to 600kb by stripping symbol table and using write() syscall over fmt (no linking required), still 590kb bigger than C equivalent

--> too much syntactic sugar to do the same task in 3 different ways, making reading someone else's code borderline impossible

and... a few gripes with the community-aspect of it...

--> community always too busy just writing wrappers for any useful library, rather than actually writing a useful library of their own. then abandoning the project after the 1.0 release, leaving the wrapper to be obsolete after like a year.

--> can't decide on toolchain design, see [this thread](#). devs make borderline subhuman iq decision, breaks the last 6 years or so of projects as a result, then tell the community that the community is just too stupid to understand how smart and exciting their "changes" are.

in short: imo language has almost no future.

the community acts like children when they get a new toy, then get bored with all their older toys when they see the big new shiny thing.

tech-wise, it does some of the things other languages already do (c, cpp, rust, ada,) but seems to do every single one of those few things worse.

maybe if they decide to develop it solely as a web-backend language and ditch any promises of actual systems development, they might have a future.

Wow, very insightful reply.

(Jun 21, 2022 03:39 PM)Faith Connors Wrote: [\[CITE\]](#) --> too much syntactic sugar to do the same task in 3 different ways, making reading someone else's code borderline impossible

Now this. This I can agree with. Although, I sorta have the same experience in Javascript...

although older JS code seems to be pretty future-proof in my experience.

I can't imagine code fully breaking because of the language updating?! That is really annoying.

I've been coding in Golang for a few weeks now before starting to really enjoy it. But, as I said... I don't really have much systems programming experience.

Rust was looking really sexy, but I went with Golang because it seemed "newer" and I've been riding the podcasts n shit.

What are your opinions on Rust vs. C? I really couldn't tell you much about how these languages "work". Guys in Gopher chat were telling me that GoLang does "threads a lot differently than C" and I have no idea what they even mean.

Faith Connors - June 21, 2022, 5:41 pm

(Jun 21, 2022 03:49 PM)Fugg Wrote: [\[CITE\]](#) What are your opinions on Rust vs. C? I really couldn't tell you much about how these languages "work". Guys in Gopher chat were telling me that GoLang does "threads a lot differently than C" and I have no idea what they even mean.

i cant say i have a ton of rust experience. not nearly as much as C or C++ at least. but i did do a 20hr video course a while ago and from the time i spent with the language, i do actually really like a lot of features (replicating oop with traits and impl{} blocks, variable shadowing, use of unsafe{}, match expressions, built-in testing,) and of course the compiler/toolchain is the most lovely thing ive ever encountered in software.

it's like, you write some code and it has a bug.

python tells you a line number and maybe highlights the thing it doesn't like.

java gives you a giant stack trace along with an error (usually nullpointerexception or indexoutofboundsexception or whatever),

c compiles fine and then segfaults when it reaches the bug,

c++ gives you a 64bit hex number to try and google for, or for runtime errors, spits out an error message somewhere in memory and you just need to know where to look for it (i'm looking at you, opengl.)

and then rust won't compile it, but it'll highlight the thing it doesn't like, give you a cookie, wrap you in a warm blanket and gives you a hug, and say "oh sweetie, it's okay. here, you just made a little mistake over here, and you probably just meant this instead :) "

im not exaggerating btw. here's an example where i "forgot" to use the enum's name, and only its value:

it just knows what im trying to do, and tells me what i need to do to fix it (e.g. turn 'On' to 'LightMsg::On')

as far as threads go, i dont really know how they're implemented in go. i can say a few words about C, but it just boils down to 'it's OS specific.'

on linux systems (or anything unix-based, like bsd or macOS) threads are implemented typically through the pthread library (POSIX threads.) it's simple, it's efficient, it does basically everything you need it to do, from scheduling to synchronicity.

on windows systems, threads are implemented in microsoft's CRT (c runtime), and they have all the functions you need as well, but the point is that C threads are typically actual OS threads, with a thread ID and everything.

from what i can gather from go, they use something called 'goroutines' which are essentially mini-threads that you can round up into one big non-blocking system/OS thread. all goroutines are non-blocking, but this implementation also means that code that uses goroutines is not necessarily using multiple cores on your computer, or spawning actual threads and creating [real parallelism](#).

what i believe is that go uses an implementation of coroutines as a compromise for threading. [here's a paper from 2000](#) on the subject, but essentially, it's the ability to leave/return a function, and then go back into it where you left off, so instead of two threads running beside each other and having idle time between thread cooperation, there's actually only one thread that goes between the two functions as needed until it's done its work.

note, i could be completely fkn wrong about the go-side of things. it's more or less an educated

guess as to how it's different. go does have proper implementation for 'real' threads if you need them, but by default goroutines aren't truly parallel.

Techcom Operator - July 16, 2022, 2:23 am

(Jun 21, 2022 05:41 PM)Faith Connors Wrote: [\[CITE\]](#) i cant say i have a ton of rust experience. not nearly as much as C or C++ at least. but i did do a 20hr video course a while ago and from the time i spent with the language, i do actually really like a lot of features (replicating oop with traits and impl{} blocks, variable shadowing, use of unsafe{}, match expressions, built-in testing,) and of course the compiler/toolchain is the most lovely thing ive ever encountered in software.

it's like, you write some code and it has a bug.
python tells you a line number and maybe highlights the thing it doesn't like.
java gives you a giant stack trace along with an error (usually nullpointerexception or indexoutofboundsexception or whatever),
c compiles fine and then segfaults when it reaches the bug,
c++ gives you a 64bit hex number to try and google for, or for runtime errors, spits out an error message somewhere in memory and you just need to know where to look for it (i'm looking at you, opengl.)

and then rust won't compile it, but it'll highlight the thing it doesn't like, give you a cookie, wrap you in a warm blanket and gives you a hug, and say "oh sweetie, it's okay. here, you just made a little mistake over here, and you probably just meant this instead :) "

im not exaggerating btw. here's an example where i "forgot" to use the enum's name, and only its value:

it just knows what im trying to do, and tells me what i need to do to fix it (e.g. turn 'On' to 'LightMsg::On')

as far as threads go, i dont really know how they're implemented in go. i can say a few words about C, but it just boils down to 'it's OS specific.'

on linux systems (or anything unix-based, like bsd or macOS) threads are implemented typically through the pthread library (POSIX threads.) it's simple, it's efficient, it does basically everything you need it to do, from scheduling to synchronicity.

on windows systems, threads are implemented in microsoft's CRT (c runtime), and they have all the functions you need as well, but the point is that C threads are typically actual OS threads, with a thread ID and everything.

from what i can gather from go, they use something called 'goroutines' which are essentially mini-threads that you can round up into one big non-blocking system/OS thread. all goroutines are non-blocking, but this implementation also means that code that uses goroutines is not necessarily using multiple cores on your computer, or spawning actual threads and creating [real parallelism](#).

what i believe is that go uses an implementation of coroutines as a compromise for threading. [here's a paper from 2000](#) on the subject, but essentially, it's the ability to leave/return a function, and then go back into it where you left off, so instead of two threads running beside each other and having idle time between thread cooperation, there's actually only one thread that goes between the two functions as needed until it's done its work.

note, i could be completely fkn wrong about the go-side of things. it's more or less an educated guess as to how it's different. go does have proper implementation for 'real' threads if you need them, but by default goroutines aren't truly parallel.

That's also really good insight on Rust. My former manager was a huge proponent of Rust and wanted to fold it into our work, but couldn't justify it to higher ups because it's another language to support and only our team would be working with it. This has probably inspired me to work with Rust for a side project and at the very least, really look deeper into it.

Related to the topic, I've used Golang for a few years at my last company and some side projects. The one thing I do like about Go is that it drilled the habit of always error checking at every assignment of a function or var so that I'm expecting the correct values. Sure, you should be doing that, but a lot of my programming experience has been catching any errors at the end and coding for happy path.

Faith Connors - July 21, 2022, 11:35 am

(Jul 16, 2022 02:23 AM)Techcom Operator Wrote: [CITE](#) The one thing I do like about Go is that it drilled the habit of always error checking at every assignment of a function or var so that I'm expecting the correct values.

then you will like the fact that this is forced in two ways in rust :)

first, the Result<T, E> enum is probably what you're familiar with from go:

the way go handles the same thing is:

Code x, err := parse_something(y)

if err != nil {

panic(err)

}

ngl, i never did get a grasp of what panic() actually does, but thats beside the point.

im assuming this is what you mean by checking at assignment.

in rust, the Result enum operates like a wrapper, which can either have the value Ok(T) (a valid return, no errors, the value of `x` above) or Err(E) (the `err` above). since this is just an enum, the data you want is either the T or E, and you need to unwrap it:

Code match some_result {

Ok(n) => return_value = n,

Err(e) => println!("{:?}", e),

}

fundamentally, it's more or less the same thing.

and yes, you can use _ to ignore data. (i.e. Ok(_) if you are only checking for an error)

the next way it's enforced is through the Option<T> enum, which is basically the same thing, but where the problem isn't an error.

say you are doing some kind of linear search on an array/vector to find a certain value, and you want to return the index.

if you find the value, the function should return the index.

but what happens when the function doesn't find it (i.e. the value isnt in the array)?

the function should return none, the same way java has nullpointerexception basically, but at least Rust forces you to actually deal with it.

Option<T> also has two values: Some(T) or None, and you check it more or less the same way you would Result.

```
Code match some_option {  
    Some(n) => index = n,  
    None => {},  
}
```

naturally, you could also make the theoretical function return a -1, but that could also be interpreted as the index of -1, or the last element of the vector.

you might also run into issues with casting between signed/unsigned integers (which, although isn't recommended and you would need to use unsafe{}, is still a way for it to go wrong.)

HFDone - July 21, 2022, 1:55 pm

same as Faith Connors says

i tried most of the known scripting and coding languages at my programming years

i can say that the most simple easiest and liked was vb net (i start with) c (native) and python (script)

Dialog - July 22, 2022, 1:49 pm

I've used Go extensively in high load systems for a handful of years now. I love its simplicity and how it has handpicked some of the good parts of OOP without the frustrating bits. The fact that it compiles self contained binaries makes it very easy to deploy and operate, and pprof offers good enough profiling that have always helped me troubleshoot things like memory and goroutine leaks on production. Most of all, however, I love how we've been able to onboard new team members without prior Go experience and saw them become productive very fast. Same can't be said for all languages. As an engineer that writes highly critical and distributed systems, I am happy with Go every day of the week.

SIM-4763645 - July 25, 2022, 3:39 am

I can't leave Go because how easy when it's come to concurrent programming

MavenGradle - July 30, 2022, 5:46 am

Go will soon be put on the 'killed by google project list'. Welcome Carbon, enjoy your stay while you can.

TerminalJester - July 30, 2022, 6:55 am

(Jun 21, 2022 02:48 PM)Fugg Wrote: [CITE](#) I just recently started playing around with GoLang. Wow. I love it. Especially with CGo... is really awesome. Programming in Go is very straight forward and easy, and most syscalls and whatnot are already in official packages. Anything else can be done with C using CGo. I haven't been doing hardly any systems programming or anything like that (mostly backend javascript web stuff for me)... and GO has been a GREAT language to get into some system functions and shit.

I haven't used CGo at all really yet. Last time I coded C/C++ was ~2013... so, needless to say... I forgot everything that I know about C.

I am having a great time using Golang to research how RunPE works and developing my own flavor.

Any Gophers out here?

I've actually just started learning it today with the app Sololearn on the App Store. So far I like and it is pretty straight forward. I'm looking forward to learning and using it!

Faith Connors - July 30, 2022, 10:01 am

(Jul 30, 2022 05:46 AM)MavenGradle Wrote: [\[CITE\]](#) Go will soon be put on the 'killed by google project list'. Welcome Carbon, enjoy your stay while you can.
sorry you need 10yrs of carbon experience to post in this thread

but honestly agree.

on a serious and slightly related note, did dart ever gain any traction as a serious language that people actually used for things or was it always and forever just a meme? i was asking a buddy of mine while discussing carbon and go a while ago and he didnt really know either.

i cant name a single product/project that unironically uses dart, nor can i name a single large body/company/internet personality that isn't google(-sponsored) that unironically endorses dart either (at least off the top of my head)

im gonna rewrite the go toolchain and stdlib in rust now, brb

MavenGradle - July 31, 2022, 8:49 am

(Jul 30, 2022 10:01 AM)Faith Connors Wrote: [\[CITE\]](#) sorry you need 10yrs of carbon experience to post in this thread

but honestly agree.

on a serious and slightly related note, did dart ever gain any traction as a serious language that people actually used for things or was it always and forever just a meme? i was asking a buddy of mine while discussing carbon and go a while ago and he didnt really know either.

i cant name a single product/project that unironically uses dart, nor can i name a single large body/company/internet personality that isn't google(-sponsored) that unironically endorses dart either (at least off the top of my head)

im gonna rewrite the go toolchain and stdlib in rust now, brb

I see where your question regarding Dart comes from, but can also quickly disprove your thoughts.

I think if I mention Flutter that you'll find that Dart does have its use cases. However, the question once again remains: for how long, before it's shelved?

<https://killedbygoogle.com/>

In addition, fuck rewriting Go's standard lib and toolchain, we do not need a garbage collector in Rust. Rust is, in my opinion where the future is at. Not Carbon, not Go, not Dart.

Fugg - August 2, 2022, 10:56 am

(Jul 31, 2022 08:49 AM)MavenGradle Wrote: [\[CITE\]](#) Rust is, in my opinion where the future is at. Not Carbon, not Go, not Dart.

Since I began this thread... I quit using Go and began learning Rust.

So, a few weeks or more of using Rust daily.

You couldn't be more right. I really enjoy the tooling and ecosystem and how everything works together... and on top of all that... the perfect type system and memory managment with ownership and borrowing.

It is really fun :D

Patience - September 4, 2022, 12:43 am

(Jun 21, 2022 03:39 PM)Faith Connors Wrote: [\[CITE\]](#)

for a systems language it seems to suck a lot of ass at programming systems lol

i tried it (and tried to shill it) around 2018 when it was at the peak of its hype (even one of my university's classes used it to teach paradigms) and honestly i just kept finding flaws over and over with it.

--> no pointer-arithmetic or anything considered 'unsafe'

--> helloworld binary size at 2mb, managed to get it down to 600kb by stripping symbol table and using write() syscall over fmt (no linking required), still 590kb bigger than C equivalent

--> too much syntactic sugar to do the same task in 3 different ways, making reading someone else's code borderline impossible

and... a few gripes with the community-aspect of it...

--> community always too busy just writing wrappers for any useful library, rather than actually writing a useful library of their own. then abandoning the project after the 1.0 release, leaving the wrapper to be obsolete after like a year.

--> can't decide on toolchain design, see [this thread](#). devs make borderline subhuman iq decision, breaks the last 6 years or so of projects as a result, then tell the community that the community is just too stupid to understand how smart and exciting their "changes" are.

in short: imo language has almost no future.

the community acts like children when they get a new toy, then get bored with all their older toys when they see the big new shiny thing.

tech-wise, it does some of the things other languages already do (c, cpp, rust, ada,) but seems to do every single one of those few things worse.

maybe if they decide to develop it solely as a web-backend language and ditch any promises of actual systems development, they might have a future.

damn he went from happy to sad :(