l-m.dev / talk

>> Read Fast, Write Fast, And Run Faster:
>> How To Maintain And Iterate With V

Tuesday, October 3rd | Lecture Theatre 123 | 6 PM - 8 PM

Who Am I?

- Liam, l-m, l-m.dev, l-m@l-m.dev
- Highschool student, Year 11
- Computer science, Physics, and Computer Graphics.
- Open source @ **V Programming Language.**
- Self taught Programmer by consequence. (I love this stuff)

- Editors: VSCode at home, Helix on the go.
- OS: **FreeBSD**, **Arch Linux**, **Alpine Linux**

Programming ‘Languages’

– **Overly proficient:** C, V, x86_64 ISA, WebAssembly (MVP spec)
– **Hobbyist:** Typescript, HTML, CSS, GLSL (shader language)
– **Looking to learn:** RISC-V ISA, Haskell

Let Me Paint A Picture.

```python
@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if message.content.startswith('have you broken yet'):
        await message.channel.send('no')

    if message.content.startswith('=split') or message.content.startswith('=dryrun'):
        chunks = str.split(message.content)
        "=split 3 jermasplit"
        if len(chunks)==3: #! if exactly 4 entries in paramaters (message)
            if is_number(chunks[1]) and len(chunks[2])<16: #! idiot proofing and string formatting
                if len(message.attachments)==1: #! check if more than one attachment or if it even has attachments
                    if any(message.attachments[0].filename.lower().endswith(image) for image in image_types): #! check if attachment is image
                        if len(chunks)==3: #! if exactly 4 entries in paramaters (message) twice?
                            if is_number(chunks[1]) and len(chunks[2])<16 and chunks[2].isalnum() and checkforascii(chunks[2]): #! idiot proofing
                                if sizecheck(message.attachments[0].size,chunks[1],chunks[1]): #! CHECK SIZE OF IMAGE


                                                    await message.channel.send(printstr)
                                                else:
                                                    await message.channel.send(sizeerror(message.attachments[0].size,chunks[1],chunks[1]))
                                                    pass
                                            else:
                                                await message.channel.send(check(message))
                                        else:
                                            await message.channel.send(check(message))
                                    else:
                                        await message.channel.send(check(message))
                                else:

                                    await message.channel.send(check(message))
                            else:
                                await message.channel.send(check(message))
                        else:
                            await message.channel.send(check(message))
```

>> Updates on the C terminal rendering engine

Apr 10, 2022 | 588 words | ~3 minute read
[C] [Graphics Programming] [ASCII] [Legacy Page]

(Legacy) Heaps of improvements! Mesh importers, depth buffer and occlusion
Read more ⟶

>> A month with C and the terminal - Part 1

Apr 4, 2022 | 698 words | ~4 minute read
[C] [Graphics Programming] [ASCII] [Legacy Page]

A (legacy) post about my first introductions to low level programming, limited graphics programming and the C language. Part 1 of a 3 part series.
Read more ⟶

>> A month with C and the terminal - Part 2

Apr 4, 2022 | 547 words | ~3 minute read
[C] [Graphics Programming] [ASCII] [Legacy Page]

Part 2, Line drawing + simple linear algebra.
Read more ⟶

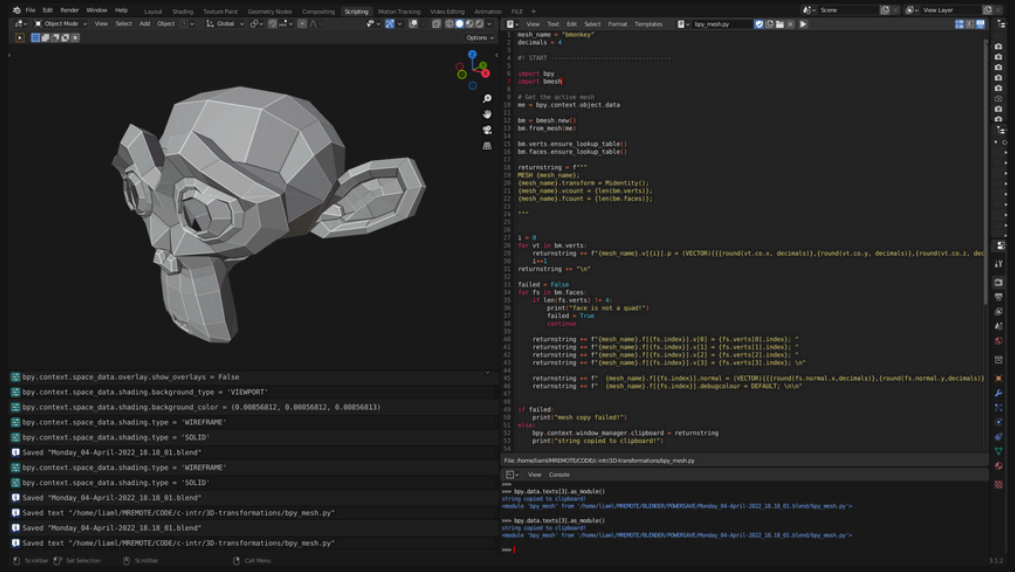
>> A month with C and the terminal - Part 3

Apr 4, 2022 | 792 words | ~4 minute read
[C] [Graphics Programming] [ASCII] [Legacy Page]

Part 3, Orthographic projection and meshes, a basic 3D engine!
Read more ⟶

>> Custom blender mesh importer

Using this custom blender script on a mesh generates code for a custom header file that can be inserted inside the main function. The generated header keeps descriptions for its vertex and face indexes plus their face normals. A rundown of how the script operates is down below.

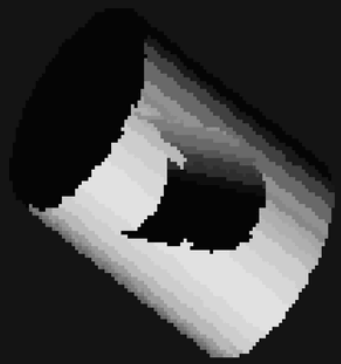


> the blender script alongside suzanne with only quad faces

```
meshes > h bmonkey.h > ...
1
2   MESH bmonkey;
3   bmonkey.transform = Midentity();
4   bmonkey.vcount = 505;
5   bmonkey.fcount = 468;
6
7   bmonkey.v[0].p = (VECTOR){9.3274,-9.6862,2.1144};
8   bmonkey.v[1].p = (VECTOR){-9.3274,-9.6862,2.1144};
9   bmonkey.v[2].p = (VECTOR){10.6598,-8.0206,0.6154};
10  bmonkey.v[3].p = (VECTOR){-10.6598,-8.0206,0.6154};
11  bmonkey.v[4].p = (VECTOR){11.6592,-5.6887,-0.2174};
12  bmonkey.v[5].p = (VECTOR){-11.6592,-5.6887,-0.2174};
13  bmonkey.v[6].p = (VECTOR){7.4952,-6.5215,-1.883};
14  bmonkey.v[7].p = (VECTOR){-7.4952,-6.5215,-1.883};
15  bmonkey.v[8].p = (VECTOR){7.4952,-8.6868,-0.7171};
16  bmonkey.v[9].p = (VECTOR){-7.4952,-8.6868,-0.7171};
17  bmonkey.v[10].p = (VECTOR){7.4952,-10.0193,1.4482};
18  bmonkey.v[11].p = (VECTOR){-7.4952,-10.0193,1.4482};
19  bmonkey.v[12].p = (VECTOR){5.8296,-10.3524,2.1144};
20  bmonkey.v[13].p = (VECTOR){-5.8296,-10.3524,2.1144};
21  bmonkey.v[14].p = (VECTOR){4.3306,-9.1865,0.6154};
22  bmonkey.v[15].p = (VECTOR){-4.3306,-9.1865,0.6154};
23  bmonkey.v[16].p = (VECTOR){3.3312,-7.1878,-0.2174};
24  bmonkey.v[17].p = (VECTOR){-3.3312,-7.1878,-0.2174};
25  bmonkey.v[18].p = (VECTOR){1.6656,-7.3543,3.78};
26  bmonkey.v[19].p = (VECTOR){-1.6656,-7.3543,3.78};
27  bmonkey.v[20].p = (VECTOR){2.9981,-9.1865,3.78};
28  bmonkey.v[21].p = (VECTOR){-2.9981,-9.1865,3.78};
29  bmonkey.v[22].p = (VECTOR){5.1634,-10.3524,3.78};
30  bmonkey.v[23].p = (VECTOR){-5.1634,-10.3524,3.78};
31  bmonkey.v[24].p = (VECTOR){5.8296,-10.3524,5.6122};
32  bmonkey.v[25].p = (VECTOR){-5.8296,-10.3524,5.6122};
33  bmonkey.v[26].p = (VECTOR){4.3306,-9.1865,6.9447};
34  bmonkey.v[27].p = (VECTOR){-4.3306,-9.1865,6.9447};
```

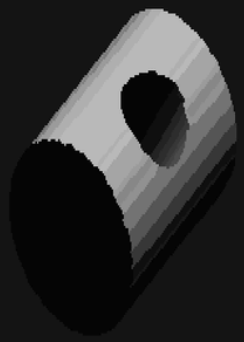
>> Working depth buffer and Self occlusion

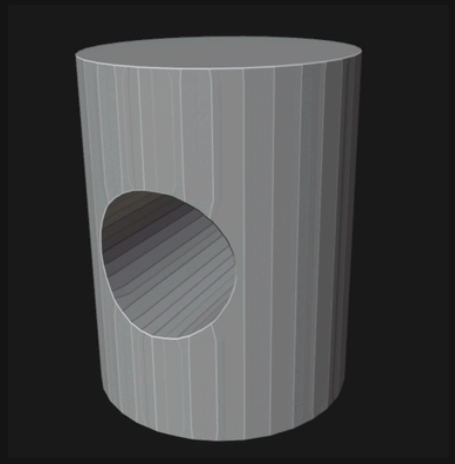
After importing a quad mesh with a hole cut out of it, I noticed something unusual. Faces that should be occluded, were showing! This was not an issue before as I rendered cubes and spheres that did not contain any holes or crevasses.



Although backface culling was implemented, faces that were facing the camera were still being shown even if they should be covered by other parts of the mesh.

Implementing a working depth buffer was long overdue. I used per face depth information to discard pixels that behind already existing pixels. Ensuring faces behind by other faces will not show up.





However?

Back And Forth

C

---

1. **OOP seemed like an Anti-Pattern to me**
   – I prefer going Data Driven.
   – Embrace Value Types.
2. **Reasonable Control Flow**
   – No exceptions, literally.
3. **“C is a universal language”**
   – Stable ABI and portable everywhere.
   – Debuggers of all kinds.
   – ASAN helps too!
4. **Static, with Concrete Types**
   – Susceptible to static analysis.
   – `-Wall -Wextra -fsanitize=address`

Python

---

1. **Get Sh*t Done Mentality**
   – Do things quick and do things now.
   – ~~Ignore the unmaintainable mess 1 month on.~~
2. **Large Ecosystem**
   – `pip` anything you want.
   – Large Standard Library.
3. **Great for Beginners**
   – INFO1110 declares it a necessary evil.
4. **Pythonic?**
   – Syntax sugar where needed.
   – Decently readable and terse code.
   – ~~Don’t buy into the cult.~~

// tldr: C programs are easier to reason about than Python programs.

// tldr: Python programs are quicker to write than C programs.

This Talk?

```
// TODO: Introduce V as an open source programming language and community.
// TODO: Teach the language to newcomers.
// TODO: Allow the audience to understand why V makes the choices it does.
// TODO: Assert V as a real competitor for writing maintainable software fast.
```

>> What Is V?

What Is V?

“Simple, fast, safe, compiled. For developing maintainable software.”

A modern programming language created by Alexander Medvednikov (Alex M) in 2018.

```v
fn main() {
    println("Hello SYNCS!")
}
```

- **The language is simple but expressive, and thus programs are maintainable.**
- Incredibly fast (debug) compilation, compile+run usually sub second.
- Production performance on par with C with zero cost C interop.
- Small compiler with zero dependencies.
- Hot code reloading, test runner, among many other features built in.
- Extremely Portable.

V Beginnings

Alex began Volt app development in 2017.

A native desktop client for major messaging services.

Volt was originally written in Go, then rewritten to C.

Go (~5 MiBs) –> C (~100 KiBs)

C development is not very productive. (Tough pill to swallow.)

In early 2018, Alex created the V language to rewrite Volt.

```
Alex M> I was writing an app in Go, but it needed a C library, CGO is slow
        and hard to debug. I also wasn't getting the C++ performance I wanted.
Alex M> So I quickly wrote a small language that translated to C and allowed
        to insert C code via `#printf("hello");`
```

V was then open sourced in June 22, 2019.

Where Do I Fit In?

I entered the community back in late 2021 looking for a programming language to use as an alternative to C and Python.

Enjoyed my time greatly. Wrote (mostly) everything in V.

Rasterisers, Ray Tracers, Ray Marchers, Software or Hardware accelerated. Programming Languages, Compilers and Interpreters. Programming Language Tools. Websites, Static Site Generators, URL Shorteners. Complex Physics Simulations. Etc..

21 long form articles on l-m.dev/cs

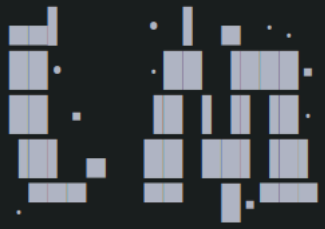
~150 short form posts on me.l-m.dev

In early 2023 I started major work on the compiler and standard library, becoming an organisation member and key community figure.

Where Do I Fit In?

- **2023 — V WebAssembly Backend and Libraries**
  — Generate WebAssembly code in memory, just `` `import wasm` ``.
  — Compile projects to WebAssembly, for the *browser* or *WASI*.

- **2023 — "me.l-m.dev"**
  — A stylistically minimal, privacy respecting, linear blogging website.
  — Zero Javascript, Zero Cookies, Zero Tracking.
  — I built this in a weekend!
- **2022 — stas**
  — A stack based compiled systems programming language.
  — 356 byte Hello World static Linux executable.
  — Leverages native system calls for Linux and FreeBSD.
  — The stas compiler is entirely selfhosted.
  — Build an identical compiler in 80ms!
  — **The 1.0 stas compiler was bootstrapped from a compiler written in V.**
- **2022 — jitcalc**
  — A calculator that evaluates expressions by creating x86_64 programs at runtime.
- **2022 — crepl**
  — Compile and execute C code on the fly as you type it.
  — Lightweight and incredibly fast alternative to *igcc*.



https://l-m.dev - © l-m.dev 2023

Welcome! - 718 total posts. [RSS]

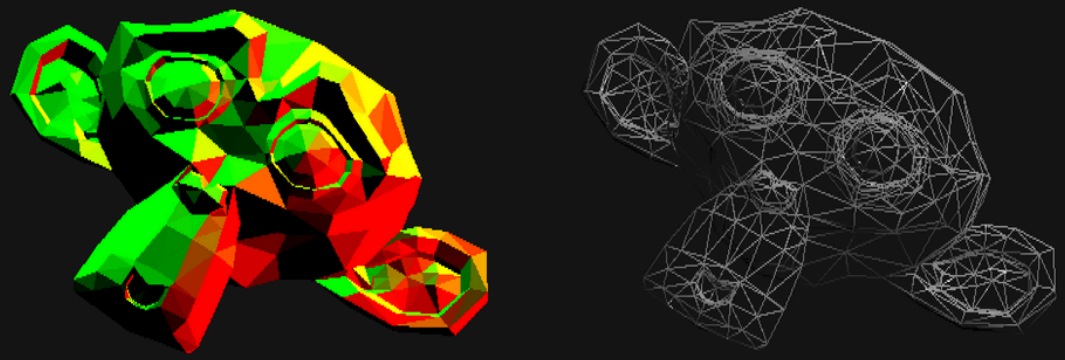
Created with <3 with V. [LATEST]

▼ Search 718 posts with 253 unique tags

[ ] cs: 415 [ ] music: 183 [x] vlang: 152
[ ] compiler: 136 [ ] c: 130 [ ] wasm: 95
[ ] lang_design: 73 [ ] web: 73
[ ] stas: 69 [ ] open_source: 69 [ ] 2d: 66
[ ] codegen: 66 [ ] 3d: 63 [ ] physics: 56
[ ] github: 54 [ ] algorithm: 50 [ ] cli: 49
[ ] optimisation: 48 [ ] self: 46 [ ] rant: 46
[ ] blender: 45 [ ] assembly: 43 [ ] emscripten: 41
[ ] sokol: 40 [ ] imgui: 36 [ ] blog: 35
[ ] hyperpop: 34 [ ] good_read: 33 [ ] vocaloid: 29
[ ] software_renderer: 28 [ ] glitchcore: 25 [ ] idea: 24
[ ] bug: 22 [ ] hatsune_miku: 22 [ ] game_engine: 22
[ ] linux: 20 [ ] osdev: 19 [ ] early_stas: 19 [ ] scene: 16
[ ] hardsurface: 15 [ ] freebsd: 13 [ ] stdlib: 13 [ ] dnb: 12
[ ] breakcore: 12 [ ] js: 11 [ ] bytecode: 11 [ ] hardware: 11
[ ] machine_girl: 11 [ ] digital_hardcore: 11 [ ] animation: 10 [ ] sh: 10
[ ] 2.5d: 10 [ ] unsafe: 10 [ ] c++: 9 [ ] dariacore: 9 [ ] pine64: 9

Instead of checking against every pixel on the whole frame, we check against every pixel contained inside those bounds. It's a key factor that keeps rasterisers fast!

>> Supporting triangles



> Face Normal Vector of each triangle and a Wireframe view of the model

I went with the decision to support triangles because any shape can be broken down into smaller triangles. Anything and everything can be represented by triangles, that's why graphics cards are so optimised for triangles and triangles only. Originally I couldn't render the Blender Suzanne since it contained a small amount of triangle faces and the rest quads. Once that was complete, any mesh that I imported using a refactored blender python importer, could be rendered.

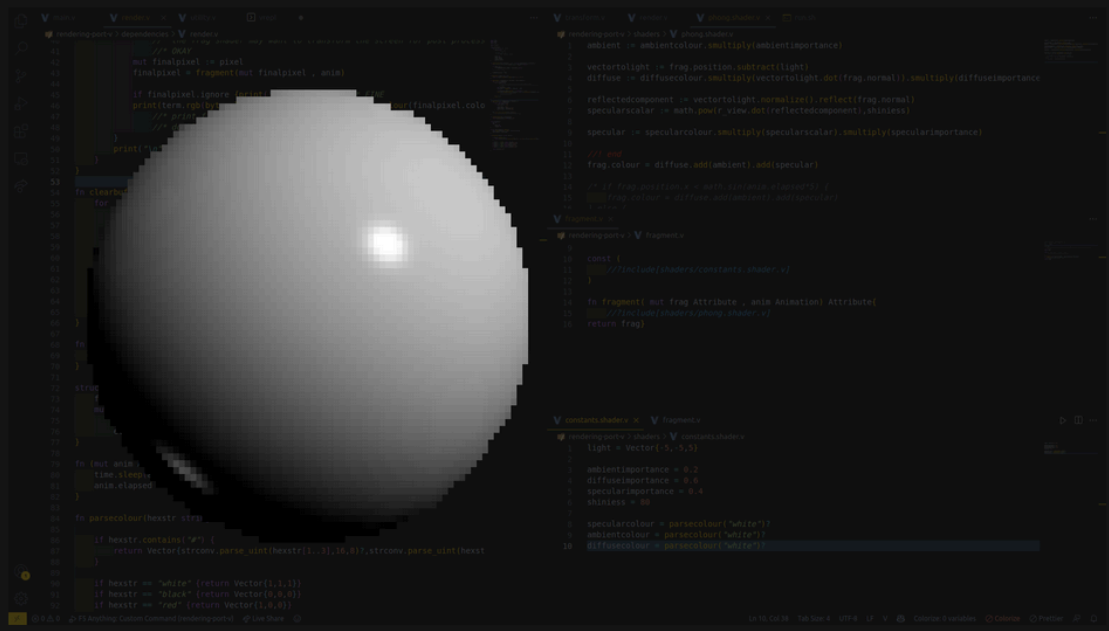
I had to change the intersection code for one that works with 4 points (quad face) to one that works with 3 (triangle). That was fairly simple though and I went on to working on vertex interpolation next.

>> The *normal* vector

> The diffuse, ambient and specular components

>> The final shaded look

```
frag.colour = diffuse.add(ambient).add(specular)
```



>> A Quick Look Into V

Compile and Run — To any Target

```v
// main.v

fn main() {
    println("Hello from l-m!")
}
```

```
$ v main.v
$ ./main
Hello from l-m!
```

```
$ v run main.v
Hello from l-m!
```

C

```
$ v main.v -o main.c
$ cc main.c -o main
# ./main
```

JS

```
$ v -b js main.v
$ node main.js
# -b js_browser
# -b js_node
# -b js_freestanding
```

NATIVE

```
$ v -b native main.v
$ wc -c main
49 # yes, 49 bytes tiny!
```

WASM (me)

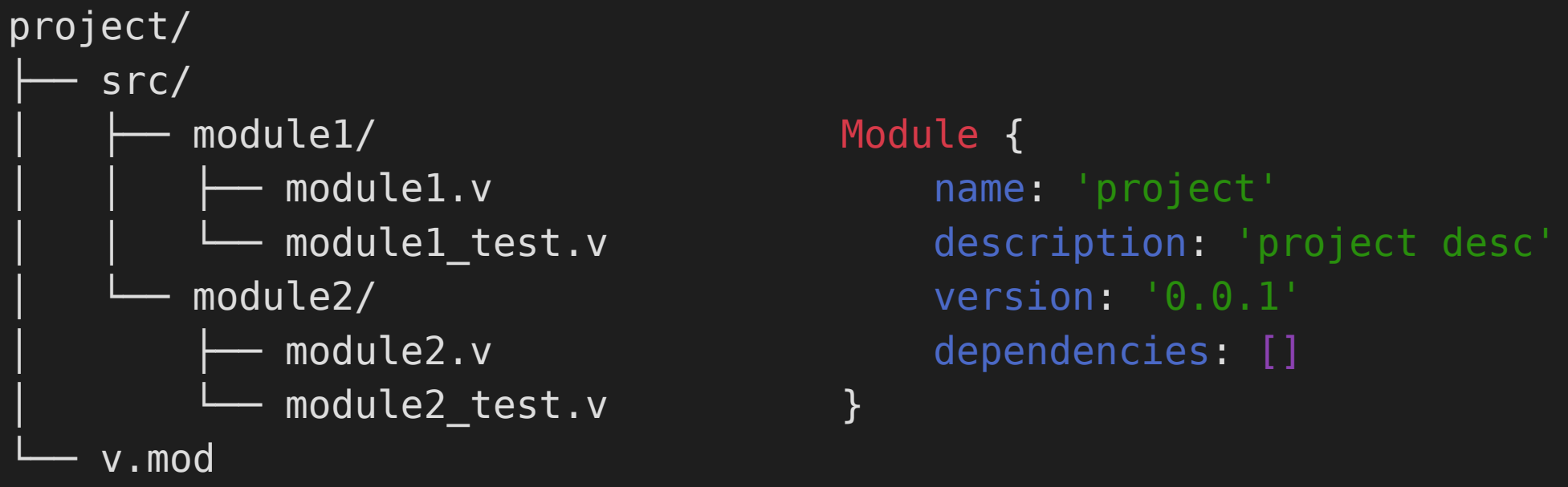
```
$ v -b wasm main.v
$ wasmer main.wasm
# implicit -os wasi
# run with wasm runtime, or
# use browser with -os browser
```

```
## Declarations - Begin


  fn main() {
      println(add(10, 15)) // prints 25!
  }

  // functions can be used before their declaration
  /* multiline comments work like this, /* and can be nested! */ */

  fn add(a int, b int) int {
      return a + b
  }
```

Declarations - Multiple Return Values

```
// swap returns two values, both ints
fn swap(a int, b int) (int, int) {
    return b, a
}
```

Modules


project/
├── src/
│ ├── module1/
│ │ ├── module1.v
│ │ └── module1_test.v
│ └── module2/
│ ├── module2.v
│ └── module2_test.v
└── v.mod
Module {
name: 'project'
description: 'project desc'
version: '0.0.1'
dependencies: []
}

Declarations - Naming Convention

- **snake_case** — Function, variable, module, and constant names
- **PascalCase** — Type names

```
// sorry! (even highlighters pick this up)
fn TestCase() {}

// allowed
fn test_case() {}
```

Declarations - Variables

Line by line:

```
fn main() {
    name := 'V'
    age := 4
}
```

```
fn main() {
    mut counter := 10
    counter = 20 + 1
}
```

Multiple on one:

```
fn main() {
    name, age := 'V', 4
}
```

Use the **`mut`** keyword to make a variable mutable, without it, reassigning is an error.

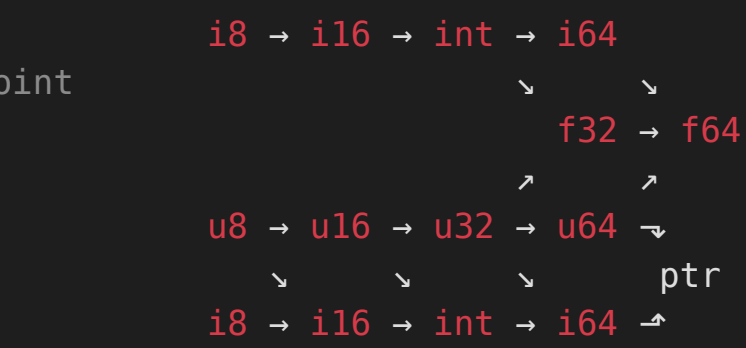
Basic Datatypes

```
i8  i16  int  i64      i128 (soon)    string rune
u8  u16  u32  u64      u128 (soon)    f32    f64  bool

isize usize // platform-dependent, pointer size
voidptr     // this one is mostly used for C interoperability
```

Literals

```
a1 := "hello"  // string type
a2 := 'hello'  // use single quotes too!
b := `A`       // rune type, Unicode code point

c := u8(10)    // u8 type, 8 bits
c := f32(10)   // f32 type
d := 3.142     // f64 type

e := false     // bool type
```

Integer Type Promotion



Printing and Strings

```
fn main() {
    name := 'l-m'
    year := 2000

    println('Hello, ${name}!')

    year_msg := 'The year is ${year + 23}\n'

    print(year_msg)

    // println, print   -> stdout
    // eprintln, eprint -> stderr
}
```

Assignments continued

```
fn swap(a int, b int) (int, int) {
    return b, a
}

fn main() {
    mut c := 105

    c, _ = swap(c, 42) // ignore with `_`

    assert c == 42
}
```

Control Flow - Conditionals

```
fn main() {
    mut a := 10

    if a > 5 {
        println('a is greater than 5')
    } else if a < 5 {
        println('a is less than 5')
    } else {
        println('a is equal to 5')
    }
}
```

```
fn main() {
    num := 23
    s := if num % 2 == 0 { 'even' } else { 'odd' }
    println(s) // odd
}
```

Control Flow - Iteration

```
// C style, mut is implied
for i := 0; i < 10; i++ {
    println(i)
}

// range iterator
for i in 0 .. 10 {
    println(i)
}
```

```
// iterate over items in array
items := ['a', 'b', 'c']

for index, item in items {
    println('${index}: ${item}')
}

// forever
for {
    println('infinite loop')
}
```

Control Flow - Break and Continue

```
fn main() {
    // print numbers 7 to 10
    outer: for i := 4; true; i++ {
        for {
            if i < 7 {
                continue outer
            } else if i > 10 {
                break outer
            } else {
                println("hit: ${i}")
                break
            }
        }
    }
}
```

Advanced Datatypes - Arrays

```v
mut nums := [1, 2, 3]

println(nums)    // [1, 2, 3]

nums[1] = 5
println(nums)    // [1, 5, 3]

println(nums[1..]) // [5, 3]
println(nums[..2]) // [1, 5]
```

```v
[1, 2, 3]           // []int
['a', 'b']          // []string

[u8(1), 88, 99, 23] // []u8
[1.0, 2.0, 3.0]     // []f64


nums := [1, 2, 3, 4, 5]
println(nums#[1..-1]) // [2, 3, 4]
```

```v
mut a := []int{len: 10000, cap: 30000, init: 3}

a << 10
a << 20

assert a.len == 10002
assert a.cap == 30000
assert a[0..2] == [3, 3]
assert a[10000..10001] == [10, 20]
```

```v
// fixed size, initialised to zero
mut fnums := [3]int{}

fnums[0] = 1
fnums[1] = 10
fnums[2] = 100

assert fnums == [1, 10, 100]! // short init
```

Advanced Datatypes - Arrays

```
mut vec := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

filtered := vec.filter(it > 2 && it < 8)
map_vec := vec.map(it * 2)

assert filtered == [3, 4, 5, 6, 7]
assert map_vec == [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
board := [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]

println(board[1][2]) // 6
```

```
mut vals := [1, 2, 3]

assert vals.pop() == 3
assert vals.any(4) == false
```

Advanced Datatypes - Maps

```
    // map with keys of type `string` and values of type `int`
    m := map[string]int{}

    // map with keys of type `string` and values of type `Foo`
    m2 := map[string]Foo{}
```

```
    mut m := map[string]int{}

    m['one'] = 1
    m['two'] = 2

    for key, val in m {
        println("key: ${key}, val: ${val}")
    }

    assert m.str() == "{'one': 1, 'two': 2}"
```

Advanced Datatypes - Structs

```
struct Person {
    age  int = 0
pub:
    name string
}

p := Person{
    name: 'Alice'
    age: 23
}

println(p.name) // Alice

println(Person{name: "Sid"}) // Person{name: "Sid", age: 0}
```

Advanced Datatypes - Structs and Privacy

```
pub struct Foo {
    a int // private immutable (default)
mut:
    b int // private mutable
    c int // (you can list multiple fields with the same access modifier)
pub:
    d int // public immutable (readonly)
pub mut:
    e int // public, but mutable only in parent module
__global:
    f int // public and mutable both inside and outside parent module
}
```

Advanced Datatypes - Methods and OOP?

```
struct Person {
    age u8 @[required] // just an attribute, required field on init
}

struct Developer {
    Person
    lang string
}

fn (p Person) is_adult() bool {
    return p.age >= 18
}

fn main() {
    dev := Developer{
        age: 52
        lang: 'V'
    }

    assert dev.is_adult()
}
```

Advanced Datatypes - Option

```
fn (r Repo) find_user_by_id(id int) ?User {
    //                              ^ Option type
    for user in r.users {
        if user.id == id {
            return user
        }
    }
    return none
    //     ^^^^ if not found, return `none`
}
```

Advanced Datatypes - Result

```
fn (r Repo) find_user_by_id(id int) !User {
    //                              ^ Result type
    for user in r.users {
        if user.id == id {
            return user
        }
    }
    return error('User ${id} not found')
    //     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ if not found, return an error
}
```

Advanced Datatypes - Propagation

```
import net.http

fn get_body(url string) !string {
    //                  ^ Result type
    resp := http.get(url)!
    //                   ^ if error, propagate it
    return resp.body
}
```

Advanced Datatypes - Unwrapping

```
r := Repo{/* ... */}

user := r.find_user_by_id(99) or {
    User{-1, 'Unknown'} // default
}

// use user
```

```
r := Repo{/* ... */}

if user := r.find_user_by_id(99) {
    // use user
} else {
    // no user
}
```

Advanced Datatypes - Interfaces

```
interface Speaker {
    speak(msg string) string
}

fn greet(s Speaker) {
    println(s.speak('Hello'))
}

struct Me {}

fn (m Me) speak(msg string) string {
    return 'Hello, ${msg}!'
}

fn main() {
    m := Me{}
    greet(m)  // Me implements Speaker
}
```

Advanced Datatypes - Interfaces

```
interface Reader {
mut:
    read(mut buf []u8) ?int
}

interface Writer {
mut:
    write(buf []u8) ?int
}

interface ReaderWriter {
    Reader
    Writer
}
```

```
struct MyRW {}

fn (mut m MyRW) read(mut buf []u8) ?int {
    // ...
}

fn (mut m MyRW) write(buf []u8) ?int {
    // ...
}
```

Advanced Datatypes - Sumtypes and Match

```
struct Empty {}

struct Node {
    value f64
    left Tree
    right Tree
}

type Tree = Empty | Node

fn sum(tree Tree) f64 {
    return match tree {
        Empty { 0 }
        Node { tree.value + sum(tree.left) + sum(tree.right) }
    }
}
```

V Tools - VPM

```
v [package_command] [param]
v install [package]
v install [--once] [--git|--hg] [url]

v install ui
v install --git https://github.com/vlang/markdown
```

---

install	Install a package from VPM.
remove	Remove a package that was installed from VPM.
search	Search for a package from VPM.
update	Update an installed package from VPM.
upgrade	Upgrade all the outdated packages.
list	List all installed packages.
outdated	Show installed packages that need updates.

V Tools - Test

```v
// add_test.v
fn test_add() {
    assert 1 + 1 == 2
}

// sub_test.v
fn test_sub() {
    assert 1 - 1 == 1 // not true
}
```

```
$ v test .
---- Testing... ----------------------------------------------------------
OK    [1/2]   178.071 ms /tmp/v/add_test.v
FAIL  [2/2]   180.654 ms /tmp/v/sub_test.v
/tmp/v/sub_test.v:2: ✗ fn test_sub
> assert 1 - 1 == 1
    Left value: 0
    Right value: 1
....
```

Memory - References

```
struct Foo {
    inner &Foo
    //    ^^^^ reference to Foo
}

fn foo(foo &Foo) {
    //     ^^^^ reference to Foo
}

// none, or &Foo
type NullableFoo = ?&Foo
```

```
struct Bas {
mut:
    num f32
}

mut value := Bas{num: 23.0}
mut value_ptr := &value

value_ptr.num = 24.0

println(value_ptr)  // &Bas{24.0}
println(*value_ptr) // Bas{24.0}
```

```
value := Bas{/* ... */}

mut value_ptr := &value  // error: `value` is immutable, ...

value_ptr := &value
value_ptr.num = 24.0     // error: `value_ptr` is immutable, ...
```

Memory - Stack and Heap

```
struct Foo {
    inner f32
}

fn return_reference() &Foo {
    // heap
    value := &Foo{inner: 23.0}

    return value
}

fn use_reference() {
    // stack
    value := &Foo{inner: 29.0}

    // use value
}
```

Globals?

```
// constant value, doesn't change
const pi_float = 3.14

struct App {
    state f64
    // ...
}

fn (mut a App) perform_action(state f64) {
    // ...
    a.state = state
}

fn (mut a App) run() {
    a.perform_action(10.0)
    a.perform_action(15.0)
    a.perform_action(pi_float)
}
```

```
// no globals required.
//
// everything happens inside App methods,
// mutating the App struct.

fn main() {
    mut app := App{}

    app.run()

    assert app.state == pi_float
}
```

What is managing all this memory?

It’s flexible:

1. Incremental Tracing GC (default)
   – Boehm GC, the most sophisticated GC present in prodution today.
   – Tunable at the compilation level.
   – **No, it won’t kill you.**
2. **Autofree**
   – Compiler assisted memory management.
3. Manual Memory management
   – It’s there, but you probably won’t need it.
4. Arena Allocation
   – V’s allocation functions will use a custom bump allocator.

- All standard library functions actually manage their own memory explicitly, turning off the GC is no problem.

Low Level Code - Vinix

```
    // must use `v -enable-globals`
    __global (
        kernel_code_seg = u16(0x28)
        gdt_pointer     GDTPointer
        gdt_entries     [11]GDTEntry
    )

-----------------------------------------------------

    pub fn interrupt_state() bool {
        mut f := u64(0)
        asm volatile amd64 {
            pushfq
            pop f
            ; =rm (f)
        }
        return f & (1 << 9) != 0
    }

-----------------------------------------------------

mut volatile cmd_ptr := unsafe { &AHCIFISh2d(&cmd_table.cfis) }

unsafe {
    C.memset(cmd_ptr, 0, sizeof(AHCIFISh2d))
}
```

Summed Up Safety

- Bounds checking.
- Built in **type safe** datatypes (arrays, maps, strings).
- No undefined values, all zero.
- No variable shadowing.
- Immutable by default.
- Explicit control flow.
- Error handling with option and result types.
- Generics and compile time reflection.
- Modular language, not OOP.
- No globals, only immutable constants.
- Opt in to unsafety with `` `unsafe {}` `` blocks.

>> Putting It All Together

Library - vweb

```v
import vweb

struct App {
    vweb.Context
}

@[get; '/users/:user']
pub fn (mut app App) user_endpoint(user string) vweb.Result {
    return app.json({
        user: 20
    })
}

pub fn (mut app App) index() vweb.Result {
    show := true
    name := 'SYNCS'
    numbers := [1, 2, 3]
    return $vweb.html() // compile time templating
}

fn main() {
    println('vweb example')
    vweb.run(&App{}, port)
}
```

```
<!-- index.html -->

@include 'header.html'

<h1>Hello @{name}!</h1>

<p>is demo?:</p>

@if show
    <p>yes</p>
@else
    <p>no</p>
@end

@for i, number in numbers
    <p>number @{i}: @{number}</p>
@end

<hr>
@include 'footer.html'
```

Library - ORM

```v
import time

[table: 'posts']
struct Post {
    id int [primary; sql: serial]
mut:
    created_at time.Time
    tags       string // space separated
    content    string
}

-----------------------------------------------

import orm

// takes an interface
fn list_posts(db orm.Connection)! {
    posts := sql db {
        select from Post order by created_at desc
    }!

    for single_post in posts {
        println(single_post)
    }
}
```

```v
import db.sqlite
import time

fn main() {
    mut db := sqlite.connect("data.sqlite")!

    // will ignore if exists
    sql db {
        create table Post
    }!

    post := Post{
        created_at: time.now()
        tags: 'self cs v'
        content: 'presentation at SYNCS! ...'
    }

    sql app.db {
        insert post into Post
    } or {
        panic('sql failed: ${err}')
    }
}
```

The Entire Ecosystem - Batteries Included

Utility

- **os** ---------- Operating System Interface
- **math** ---------- Math Operations
- **json** ---------- JSON Generic Serialisation and Deserialisation
- **encoding** ---------- Various Text and Binary Encodings
- **crypto** ---------- Cryptography

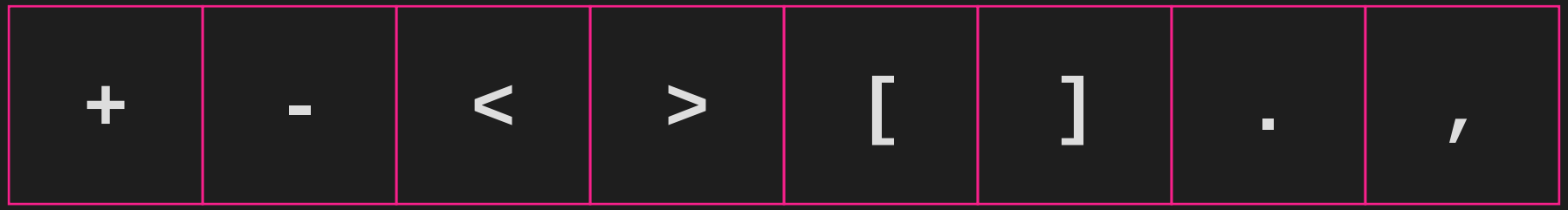
Web

- **net.websocket** ---------- Websocket Client and Server
- **net.http** ---------- HTTP Serve and Requests
- **net.html** ---------- HTML DOM Manipulation and Parser
- **vweb** ---------- V Web Framework
- **picoev** ---------- Lightning Fast Event Loop

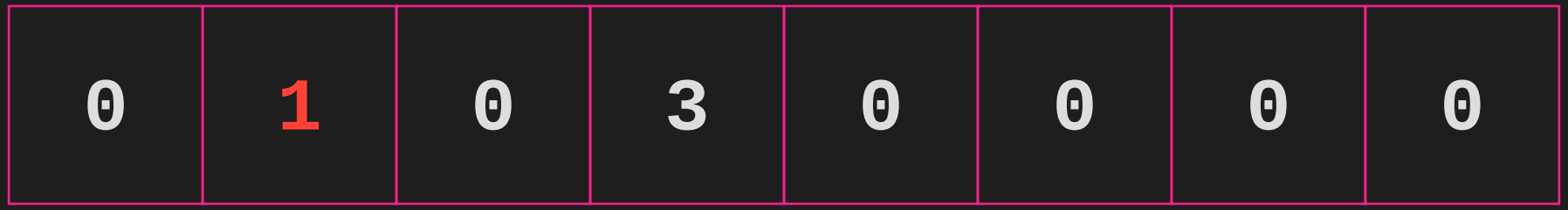
Graphic Libraries, Games, UI Frameworks

- **gg** ---------- Simple Graphics
- **sokol** ---------- Portable OpenGL API
- **ui** ---------- V UI Framework

Compiler Creation - BrainF*ck



Compiler Creation - BrainF*ck




```
>++>>+++<<-
```

```
+++++[->+<]>
     ^    !^
```

while (cell != 0) repeat between braces

Compiler Creation - 'import wasm'

```
import wasm
import os

fn main() {
    mut m := wasm.Module{}
    mut func := m.new_function('add', [.i32_t, .i32_t], [.i32_t])
    {
        func.local_get(0) // | local.get 0
        func.local_get(1) // | local.get 1
        func.add(.i32_t)  // | i32.add
    }
    m.commit(func, true) // `export: true`

    mod := m.compile() // []u8

    os.write_file_array('add.wasm', mod)!
}
```

Compiler Creation - 'import wasm'

```
import os
import wasm

fn main() {
    bf_expr := os.args[1] or {
        println("Usage: bf <expr>")
        return
    }
    mut m := wasm.Module{}

    // [accepts] -> [returns], don't worry about the implementation...
    m.new_function_import('wasi_unstable', 'fd_write', [.i32_t, .i32_t, .i32_t, .i32_t], [.i32_t])
    m.new_function_import('wasi_unstable', 'fd_read', [.i32_t, .i32_t, .i32_t, .i32_t], [.i32_t])
    m.assign_memory('memory', true, 2, none)

    mut start := m.new_function('_start', [], [])
    {
        generate_code(mut start, bf_expr)
    }
    m.commit(start, true)

    bytes := m.compile()
    os.write_file_array('bf.wasm', bytes)!
}
```

Compiler Creation - 'import wasm'

```
fn generate_code(mut start wasm.Function, bf_expr string) {
    sp := start.new_local_named(.i32_t, 'sp')

    mut loop_labels := []wasm.LabelIndex{}
    mut block_labels := []wasm.LabelIndex{}

    for ch in bf_expr {
        match ch {
            `>` {/* ... */}        // increment pointer
            `<` {/* ... */}        // decrement pointer
            `+` {/* ... */}        // increment value at pointer
            `-` {/* ... */}        // decrement value at pointer
            `.` {/* ... */}        // print ASCII character at pointer
            `,` {/* ... */}        // read ASCII character at pointer
            `[` {/* ... */}        // open loop
            `]` {/* ... */}        // close loop
            else {}                // comments, ignore
        }
    }
}
```

Compiler Creation - 'import wasm'

```
`>` {
    // sp = sp + 1
    start.local_get(sp)
    start.i32_const(1)
    start.add(.i32_t)
    start.local_set(sp)
}
`<` {
    // sp = sp - 1
    start.local_get(sp)
    start.i32_const(1)
    start.sub(.i32_t)
    start.local_set(sp)
}
```

```
`+` {
    // *sp = *sp + 1
    start.local_get(sp)
    {
        start.local_get(sp)
        start.load8(.i32_t, false, 0, 0)
        start.i32_const(1)
        start.add(.i32_t)
    }
    start.store8(.i32_t, 0, 0)
}
`-` {
    // *sp = *sp - 1
    start.local_get(sp)
    {
        start.local_get(sp)
        start.load8(.i32_t, false, 0, 0)
        start.i32_const(1)
        start.sub(.i32_t)
    }
    start.store8(.i32_t, 0, 0)
}
```

Compiler Creation - 'import wasm'

```
`.` {
    generate_ciovec(mut start, sp)

    // stdout, *iovs, iovs_len, *nwritten
    start.i32_const(1)
    start.i32_const(runtime_page)
    start.i32_const(1)
    start.i32_const(runtime_page + 1024)
    start.call_import('wasi_unstable', 'fd_write')
    start.drop()
}
`,` {
    generate_ciovec(mut start, sp)

    // stdin, *iovs, iovs_len, *nwritten
    start.i32_const(0)
    start.i32_const(runtime_page)
    start.i32_const(1)
    start.i32_const(runtime_page + 1024)
    start.call_import('wasi_unstable', 'fd_read')
    start.drop()
}
```

```
`[` {
    block_lbl := start.c_block([], [])
    loop_lbl := start.c_loop([], [])
    {
        // *sp == 0 ? jmp :block_lbl
        start.local_get(sp)
        start.load8(.i32_t, false, 0, 0)
        start.eqz(.i32_t)
        start.c_br_if(block_lbl)
    }
    loop_labels << loop_lbl
    block_labels << block_lbl
}
`]` {
    loop_lbl := loop_labels.pop()
    start.c_br(loop_lbl) // jump back to top
    start.c_end(loop_lbl)
    start.c_end(block_labels.pop())
}
```

Compiler Creation - 'import wasm'

```
cd where/is/your/v
cd examples/wasm_codegen
v run bf_compiler.v "+[>,.<]"  # simple "cat" program
wasmtime bf.wasm               # execute
```

>> Finishing Up

The **culture** at V

QnA - The End

https://l-m.dev/talk

GitHub - **l1mey112** | Email - **l-m@l-m.dev**

Entire Presentation Written With **Typst.**