# Chapter 13 – vlib – V’s standard library

## 13.1 Overview of the standard library

The V-distribution contains over 50 standard built-in modules for common functionality, like math, os, etc… as a whole designated as the *standard library*. This vlib is for the most part written in V itself, but still depends heavily upon C interop, in particular for some low level routines. It is intended to be comprehensivelike the standard library in Go, not bare-bones like that in Rust.

The API in all modules (included package os) is the same for all systems (Windows, Linux, ...).

Vlib modules are imported in your program via the import statement (see ch. 11).

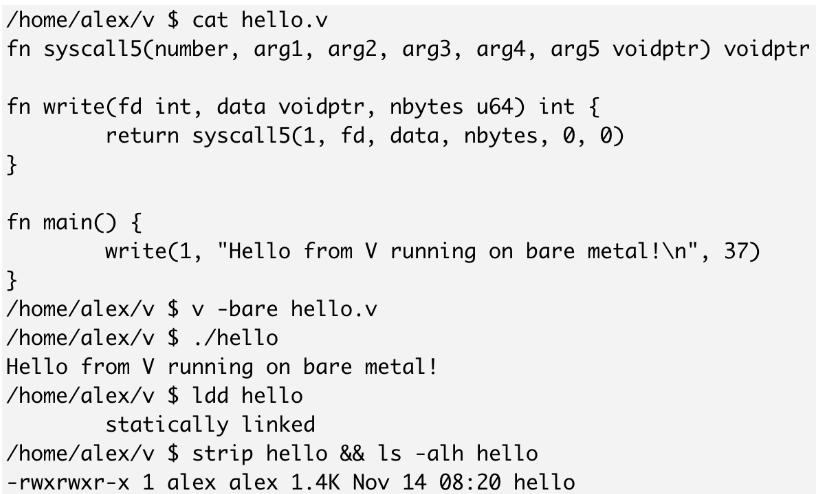
The vlib folder contains only the V source code, comprising some 2 Mb. (??) When your app is compiled, the relevant parts of Vlib are compiled with it each time a compilation is done.

The C dependency: V is a language that builds upon C to make it fast and cross-platform. V stdlib uses *libc* for some functions, in particular on non-Linux systems, since it's their only official API. Everything that uses unstable and/or private syscalls is going to keep using libc, things like sprintf will be replaced with native V code. Removing libc completely would only work on Linux.

On libc dependency: <https://github.com/vlang/v/wiki/On-libc-usage-(early-draft)>

# V now runs on bare metal (freestanding) (Nov 16 2019): <https://github.com/vlang/v/issues/2761>

No libc or vlib is needed. From now on V can be used for embedded, kernel development, etc.



Works on Linux!

Vlib modules:

Docs can be obtained with the command: v doc module

For example: v doc os

V doc http

(?? Nov: doesn’t work on Windows: V panic: failed to open file "C:\Users\CVO\.vmodules/vlib/os.vh"

Doesn’t work on Linux either)

Here is an overview of the most important vlib modules. In the following sections, we elaborate on some of them,discussing concrete programs and applications.

arrays: see ch 7

benchmark: contains simple methods to benchmark the execution of V code, see § 13.??

bitfield: manipulating arrays of bits

builtin: contains v's predeclared functions and types (including strings, arrays, maps); this is always by default imported in a V program, also in the REPL.

[compiler](https://vlang.io/pkg/compiler) : V compiler as a module

[crypto](https://vlang.io/pkg/crypto):

[aes](https://vlang.io/pkg/crypto.aes): AES encryption (formerly Rijndael), as defined in U.S. Federal Information Processing Standards Publication 197

[cipher](https://vlang.io/pkg/crypto.cipher): Standard block cipher modes that can be wrapped around low-level block cipher implementations

[encoding](https://vlang.io/pkg/base64):

[base64](https://vlang.io/pkg/base64): Encoding and decoding of base64 as defined in RFC 4648

[binary](https://vlang.io/pkg/encoding.binary): Simple translation between numbers and byte sequences and encoding and decoding of varints

[csv](https://vlang.io/pkg/encoding.csv): Reads and writes comma-separated values (CSV) files

gg: Hardware accelerated Graphics library using OpenGL (DirectX, Vulkan, Metal support soon)

gl: OpenGL wrapper

gx: Constants and helpers for drawing

http: contains functionality for parsing HTTP requests/replies, provides an extensible HTTP server and a basic client;

json: JSON-support is built-in: encoding and decoding of JSON as defined in RFC 7159, see § 13.??

log: contains logging facilities

math: contains the basic mathematical constants and functions.

For example: *greatest\_common\_divisor.v*

import math

fn main() {

  res := **math.gcd**(4585, 1589)

  println(res) // 7

}

For some common mathematical constants and operations, see *math\_op.v*

To see an example of calculations at work: see *nbody.v, spectral.v*

os: gives a platform-independent interface to operating-system functionality

mysql:

pg:

sqlite: connecting to databases of various types, see $ 13.??

strings: contains functions for manipulating and processing strings.

rand: contains routines for generating pseudo-random numbers

runtime: For example (see runtime.v):

import runtime

println(runtime.**nr\_cpus()**) // 4

time:  contains basic functionalities for working with times and dates (see $4.??)

ui: cross-platform ui library

vweb: v’s web framework

## 13.2 os module

The module’s design is Unix-like; it hides the differences between various operating systems to give a consistent view of files and other OS-objects.

Some of the example code can be found in os\_examples.v, unless otherwise mentioned.

See § 5.1 for detecting the operating system with $if.

Command-line arguments: these are given by os.args, which is of type []string;

for example: os.args[1] is the first command-line parameter.

See os\_args.v:

import os

fn main() {

    for ix, arg in os.args {

      println('$ix: $arg')

    }

}

/\* When called as:  os\_args A 1 Hello          This prints out:

E:\Vlang\The\_Way\_to\_V\Chapter\_9\_Modules>os\_args A 1 Hello

0: os\_args

1: A

2: 1

3: Hello

\*/

The first element os.args[0] is the program name without the .v extension. To skip this first element, you would do:

    for i := 1; i < os.args.len; i++ {

        println(os.args[i])

    }

A simple example of parsing options can be found in option\_parser() (see Vcasino/vcasino.v)

Exercise: Use a stack to program a primitive console calculator, usign reverse polish notation (see *calc1.v* and *calc.v*)

Using the **flag** module for parsing command line arguments: ??

Get environment variables: *os\_env.v*

import os

fn main() {

  home := **os.getenv**('PATH')

  println(home)

}

Detecting some properties of the OS: *os\_props.v*

os.**executable**() - a cross platform function that returns full path to current executable.

Getting input from the terminal:

In this example, input from the user is collected via the os.get\_line function: this retrieves a single line from the console - or in other words an input terminated by a single [ENTER key](https://en.wikipedia.org/wiki/Enter_key).

see *os\_get\_line.v*

Here is a guessing example guessing\_game.v:

// Guess a random number between 1 and 100

import (

  os    // contains the I/O functions

  rand  // to generate a random number to guess

  time  // to seed the random number generating algorithm

)

const (

    max\_nr\_guess = 10

)

fn main() {

// 1 - Generate random number:

    t := time.now()

    s := t.calc\_unix()

    rand.seed(s) // calls C.srand(s)

    secret := rand.next(100) // random number from 1 to 100 for the user to guess

    mut nr\_guess := 0

// 2 - Game loop:

    for {

        println('Please guess a number from 1-100 or stop(S) and press <Enter>')

      guess := **os.get\_line().trim\_space()**          // input from user

        if guess == 'S' {

            println('Game stopped')

            return

        }

        nr\_guess += 1

        if nr\_guess > max\_nr\_guess {

            println('Sorry, too much guessing, the secret number was: $secret')

            println('Game over')

            return

        }

        println('You guessed $guess on turn $nr\_guess')

      iguess := **guess.int()** // convert guess to integer

        if iguess > secret {

            println('Too big!')

        } else if iguess < secret {

            println('Too small!')

        } else if iguess == secret {

            println('Congratulations! You guessed the secret number $secret in $nr\_guess turns!')

            return                      // end loop

        } else {

            print('Incorrect! ')

        }

    }

    println('Thanks for playing the game!')

}

Another example: see *Vcasino/vcasino.v*

For another input reading routine, containing additional checks to test for an empty line, and to test that the string is a number:, function get\_bet\_nbr().

When the input is a string: line := os.get\_line().trim\_space().to\_lower()

Working with files:

The os module uses optionals in all functions that return `File`.

A file’s size in bytes is given by:

os.**file\_size**(file\_name)

Check whether a file exists with: os.**file\_exists**(path)

*(see word\_counter\_book.v)*

How to traverse a directory and get its contents?  os.ls('.') returns ?[]string

On Linux: os.ls(pwd)

Opening a file:

fn read\_log() {

**f := os.open(****'log.txt')**

**defer { f.close() }**

...

if !ok {

// defer statement will be called here, the file will be closed

return

}

...

// defer statement will be called here, the file will be closed

}

When opening a file explicitly, this is followed immediately by a defer block to close the file in all cases at the end of the function?

Reading in a file follows the general pattern:

import os

text := **os.read\_file**(path)**or{**

**eprintln(err)**

**}**

read\_file() reads the whole contents of the file into a string text, signature: os.read\_file(path) string

(Example: see word\_counter.v)

How to read in a file line by line:

Use the method read\_lines, which has the signature: os.read\_lines(path) ?[]string

lines := os.**read\_lines**(filepath) or {

println(**'**File not found!**')**

return

}

Then you can iterate over the lines as follows: for line in lines { … }

Or pick a random line with: lines[rand.next(lines.len)]

For an example, see hangman/hangman.v

An alternative: contents := os\_.read\_file(path) or { }

lines := contens.split\_into\_lines()

(see faker module ch 14)

Writing to a file: see *file\_write.v*

import os

fn test\_file\_creation() {

    file\_name := './new\_file.txt'

    content := 'text'

**os.write\_file**(file\_name, content)

    assert content.len == os.file\_size(file\_name)

**os.rm**(file\_name)

}

test\_file\_creation()

This is done by using the os.write\_file function, which overwrites an existing file with that name.

os.rm deletes the file.

Executing commands: (see *os\_exec.v*)

This is done with the os.system function, which takes a command string:

import os

fn run(cmd string) {

  if **os.system(cmd)** != 0 {

    panic('"$cmd" failed')

  }

}

// on Windows:

run('dir')

run('copy a b') // => The system cannot find the file specified.

// on nix\*:

//execute a command and fetch result string.

os.ls('pwd')

## 13.3 json module - Decoding and encoding JSON.

JavaScript Object Notation (JSON) is a lightweight data-interchange format that is easy for humans to read and write. Furthermore, equally simple for machines to generate and/or parse. JSON is completely language agnostic and that's why it's the ideal interchange format (to read more about JSON visit: [json.org](http://json.org/)).

Support for the very popular JSON format is built-in in V.

V generates code for json encoding and decoding at compile-time. No reflection is used, which results in much better performance. V doesn't need runtime information to decode JSON, because it achieves it via codegen. All information about types is known at compilation time.

The function json.decode(Struct, data) can translate the JSON string data to the fields of struct Struct, provided they have the same names and type. The first argument of the json.decode function is the type to decode to. The second argument is the JSON string. (?? Make a little sketch). The function json.encode(Struct) generates a JSON string for serialization or communication; it takes a Struct value and converts it to a JSON string.

Here are some simple examples:

Listing 13.2 json.v:

import json

struct User {

      name string

      age  int

      last\_name string [json:lastName]

}

fn main() {

      data := '{ "name": "Frodo", "lastName": "Baggins", "age": 25 }' // JSON string

      user := json.decode(User, data) or {

              eprintln('Failed to decode json-string $data')

              return

      }

      println(user.name)

      println(user.last\_name)

      println(user.age)

      // JSON string with an array of struct values

      customers\_string := '[{ "first\_name": "Vitor", "last\_name": "Oliveira", "hometown": "Rio de Janeiro" }, { "first\_name": "Don", "last\_name": "Nisnoni", "hometown": "Kupang" }]'

      customers := **json.decode([]Customer, customers\_string)** or {

            eprintln('Failed to parse json')

            return

      }

      // Print the list of customers

      for customer in customers {

            println('$customer.first\_name $customer.last\_name: $customer.hometown')

      }

// Generate a JSON string:

      customer := Customer{first\_name: "Vitor" last\_name: "Oliveira" hometown: "Rio de Janeiro"}

      encoded\_json := json.encode(customer)

      println(encoded\_json)

      expected := '{"first\_name":"Vitor","last\_name":"Oliveira","hometown":"Rio de Janeiro"}'

      assert encoded\_json == expected

     // back and forth:

      customer2 := json.decode(Customer, encoded\_json) or {

            eprintln('Failed to parse json')

            return

      }

      encoded\_json2 := json.encode(customer2)

      println(encoded\_json2)

      assert(encoded\_json == encoded\_json2)

}

/\* Output:

Frodo

Baggins

25

Vitor Oliveira: Rio de Janeiro

Don Nisnoni: Kupang

{"first\_name":"Vitor","last\_name":"Oliveira","hometown":"Rio de Janeiro"}

{"first\_name":"Vitor","last\_name":"Oliveira","hometown":"Rio de Janeiro"}

\*/

If the field name is different in the JSON file, it can be specified: custom json field names can be added between [json: ]:

struct User { last\_name string **[json:lastName]** }

Use the `skip` attribute in the struct description to skip certain fields:      foo Foo [skip]

You can also have a raw JSON field, whose content is taken literally: point string **[raw]**

This is used in the following example *json2.v* (we use asserts to test the values instead of printing them out):

import json

struct User {

  age         int

  nums        []int

  last\_name   string  [json:lastName]

  is\_registered bool  [json:IsRegistered]

  typ int  [json:'type']

}

struct Color {

  space string

  point string [raw]

}

fn parse\_user() {

  s := '{"age": 10, "nums": [1,2,3], "type": 0, "lastName": "Johnson", "IsRegistered": true}'

  u := json.decode(User, s) or {

    exit(1)

  }

  assert u.age == 10

  assert u.last\_name == 'Johnson'

  assert u.is\_registered == true

  assert u.nums.len == 3

  assert u.nums[0] == 1

  assert u.nums[1] == 2

  assert u.nums[2] == 3

  assert u.typ == 0

}

fn encode\_user() {

  usr := User{ age: 10, nums: [1,2,3], last\_name: 'Johnson', is\_registered: true, typ: 0}

  expected := '{"age":10,"nums":[1,2,3],"lastName":"Johnson","IsRegistered":true,"type":0}'

  out := **json.encode(usr)**

  assert out == expected

}

fn raw\_json\_field() {

    color := json.decode(Color, '{"space": "YCbCr", "point": {"Y": 123}}') or {

        println('Error while decoding color!')

        return

    }

    assert color.point == '{"Y":123}'

    assert color.space == 'YCbCr'

}

parse\_user()

encode\_user()

raw\_json\_field()

Here is a more concrete example, reading in a file and decoding the contents as json: *json\_file.v*

import os

import json

struct Extract {

        origin string

        target\_language string

        category int

        source string

        source\_language string

        target string

    }

fn main() {

    file := os.read\_file('weblate-memory.json') or {

        eprintln('Error opening file!')

        return

    }

    users:= json.decode([]Extract, file) or {

        eprintln('Failed to decode json')

        return

    }

    for u in users {

      println(u.source)

      println(u.target)

    }

}

/\* Output:

Card callback failure

କାର୍ଡ଼ କଲବ୍ୟାକ ଅସଫଳ

Sink callback failure

ସିଙ୍କ କଲବ୍ୟାକ ଅସଫଳ

\*/

See also § 13.4 get\_albums.v

## 13.4 http module

Example 1: *links\_scraper.v*

**import http**

fn main() {

  html := **http.get\_text(**'https://news.ycombinator.com')   (1)

  mut pos := 0

  for {

    pos = **html.index\_after**('https://', pos + 1) (2)

    if pos == -1 {

      break

    }

    end := html.index\_after('"', pos)

    println(**html[pos..end]**) (3)

  }

}

The function get\_text(URL) in line (1) gets the html text of the URL in one big string html.

(?? When URL is wrong, an error is printed:

Error 10057 sending data to server (1)

Error performing handshake

Not sure where this comes from)

In line (2) during an infinite loop, all http links are searched and copied out in line (3) to be printed.

The loop stops when no more http link could be found.

/\* Example output:

https://news.ycombinator.com

https://news.bloomberglaw.com/privacy-and-data-security/facebook-google-donate-heavily-to-privacy-advocacy-groups

https://www.intel.com/content/www/us/en/support/topics/idsa-cip.html#CollectedData

https://www.vogons.org/viewtopic.php?f=46&amp;t=69184

https://blog.floydhub.com/a-pirates-guide-to-accuracy-precision-recall-and-other-scores/

https://arstechnica.com/tech-policy/2019/11/google-search-results-have-more-human-help-than-you-think-report-finds/

https://en.wikipedia.org/wiki/Crinkle\_crankle\_wall

https://ahajournals.org/doi/full/10.1161/circulationaha.111.061770

https://github.com/norvig/paip-lisp/issues/10

https://www.1843magazine.com/culture/look-closer/hokusai-old-man-crazy-to-paint

https://github.com/khstangherlin/monker

https://www.scmp.com/lifestyle/food-drink/article/3037576/highs-and-lows-lowell-cafe-americas-first-cannabis-cafe-where

https://github.com/dosycorp/browsergap.ce

https://medium.com/through-the-looking-glass/colossal-holograms-b7f86f5925bd

https://www.nytimes.com/2019/11/14/science/stars-black-hole-milky-way.html

https://www.joshmcguigan.com/blog/understanding-serde/

https://www.currentaffairs.org/2019/11/the-innocent-pleasure-of-trespassing

https://www.collectorsweekly.com/articles/how-boomboxes-got-so-badass/

https://littlemountainman.github.io/2019/11/27/selfdrivingfun/

https://www.scottaaronson.com/blog/?p=4414

https://www.pnas.org/content/early/2019/11/11/1907883116

https://www.reuters.com/article/us-interpol-encryption-exclusive/exclusive-interpol-plans-to-condemn-encryption-spread-citing-predators-sources-say-idUSKBN1XR0S7

https://www.wbur.org/hereandnow/2019/11/15/amtrak-profit-train-ceo-richard-anderson

https://www.shacknews.com/article/114982/world-on-fire-the-oral-history-of-fallout-and-fallout-2

https://blog.benjojo.co.uk/post/userspace-usb-drivers

https://www.reuters.com/article/us-apple-germany-apple-pay/apple-warns-of-risks-from-german-law-to-open-up-mobile-payments-idUSKBN1XP16M

https://www.bfilipek.com/2019/11/perfguidecpu.html

https://github.com/HackerNews/API

\*/

Example 2: *get\_albums.v*

import (

  http

  json

)

struct Album {

  user\_id int

  id int

  title string

}

fn get\_albums() []Album {

  response := http.**get**('https://jsonplaceholder.typicode.com/albums') or {

    panic("Couldn't find the albums page" )

  }

  parsed\_albums := json.decode([]Album, response.text) or { return [] }

  return parsed\_albums

}

albums := get\_albums()

for album in albums { println(album) }

/\* Output:

{

  user\_id: 0

  id: 1

  title: quidem molestiae enim

}

{

  user\_id: 0

  id: 2

  title: sunt qui excepturi placeat culpa

}

{

  user\_id: 0

  id: 3

  title: omnis laborum odio

}

{

  user\_id: 0

  id: 4

  title: non esse culpa molestiae omnis sed optio

}

...

\*/

See also § 13.7 news\_fetcher.v

## 13.5 log module

Example:

import log

fn main(){

    mut l := log.Log{log.INFO, 'terminal'}

    l.info('info')

    l.warn('warn')

    l.error('error')

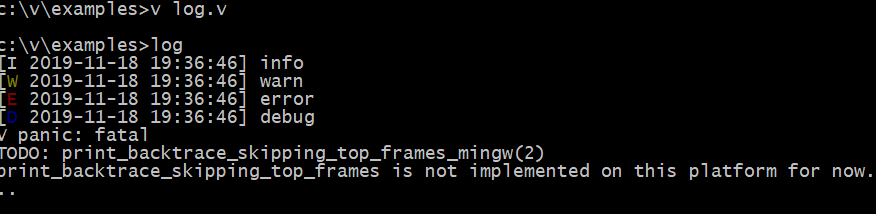
    l.debug('no debug')

    l.set\_level(log.DEBUG)

    l.debug('debug')

    l.fatal('fatal')

}



?? execute on Linux

## 13.6 term module

This allows manipulation of the terminal output properties, see *terminal\_control.v*

import term

fn sleeping\_line(x,y,size int, ch string) {

  mut i := 0

  for i < size {

    term.set\_cursor\_position(x+i,y)

    print(term.bold(term.yellow(ch)))

    i++

  }

}

fn standing\_line(x,y,size int, ch string) {

  mut i := 0

  for i < size {

    term.set\_cursor\_position(x,y+i)

    print(term.bold(term.yellow(ch)))

    i++

  }

}

term.erase\_clear()

sleeping\_line(5,5,5,'\*')

standing\_line(5,5,5,'\*')

sleeping\_line(5,10,5,'\*')

standing\_line(9,5,5,'\*')

term.cursor\_down(5)

print('\n')

println(term.bold(term.red('It Worked!')))

Output:



## 13.7 sync module

The following example *news\_fetcher.v* is taken from V’s examples. The program shows how texts in json format are retrieved from the internet and demonstrates the use of locking and the concept of a WaitGroup from the sync module, in order to synchronize 4 coroutines.

import http

import json

import sync

const (

  NR\_THREADS = 4

)

struct Story {

  title string

  url string

}

struct Fetcher {

mut:

  mu      &sync.Mutex

  ids     []int

  cursor  int

  wg      &sync.WaitGroup

}

fn (f mut Fetcher) fetch() {

  for {

    if f.cursor >= f.ids.len {

      return

    }

    id := f.ids[f.cursor]

**f.mu.lock()** (5)

    f.cursor++

**f.mu.unlock()**

    cursor := f.cursor

    resp := http.**get**('https://hacker-news.firebaseio.com/v0/item/${id}.json') or { (6)

      println('failed to fetch data from /v0/item/${id}.json')

      exit(1)

    }

    story := json.decode(Story, resp.text) or { (7)

      println('failed to decode a story')

      exit(1)

    }

    println('#$cursor) $story.title | $story.url')

    f.wg.done()

  }

}

// Fetches top HN stories in 4 coroutines

fn main() {

  resp := http.get('https://hacker-news.firebaseio.com/v0/topstories.json') or { (1)

    println('failed to fetch data from /v0/topstories.json')

    return

  }

  mut ids := json.decode([]int, resp.text) or { (2)

    println('failed to decode topstories.json')

    return

  }

  if ids.len > 10 {

    // ids = ids[:10]

    mut tmp := [0].repeat(10)

    for i := 0 ; i < 10 ; i++ {

      tmp[i] = ids[i]

    }

    ids = tmp

  }

**wg := sync.new\_waitgroup()**

**mtx := sync.new\_mutex()**

  mut fetcher := &Fetcher{ids: ids}

  fetcher.mu = &mtx

  fetcher.wg = &wg

  fetcher.wg.add(ids.len)

  for i := 0; i < NR\_THREADS; i++ { (3)

**go fetcher.fetch()**

  }

**fetcher.wg.wait()** (4)

}

/\* Output:

#1) The cognitive costs of air pollution | https://patrickcollison.com/pollution

#2) Your throat hurts, your brain hurts: the life of the audiobook star | https://www.theguardian.com/books/2019/nov/16/throat-hurts-brain-hurts-secret-life-of-audiobook-stars-tim-dowling

#4) How Containers Work: Overlayfs | https://jvns.ca/blog/2019/11/18/how-containers-work--overlayfs/

#3) Hacker Publishes 2TB of Data from Cayman National Bank | https://twitter.com/DDoSecrets/status/1195899716653010945

#5) MacBook Pro 16" 2019 Teardown | https://www.ifixit.com/Teardown/MacBook+Pro+16-Inch+2019+Teardown/128106

#7) Empathic concern does not reduce partisan animosity: study | https://www.wired.com/story/empathy-is-tearing-us-apart/

#8) Sourcetrail, interactive source explorer, is now free and open-source | https://www.sourcetrail.com/blog/open\_source/

#6) Office noise bothers some people more than others | https://www.bbc.com/worklife/article/20191115-office-noise-acceptable-levels-personality-type

#9) A Dead-Simple Web Stack in Haskell | https://williamyaoh.com/posts/2019-11-16-a-dead-simple-web-stack.html

#10) Music Generates Feelings That Are Only Weakly Bound to the Music | https://whatismusic.info/blog/MusicGeneratesFeelingsThatAreOnlyWeaklyBoundToTheMusic.html

\*/

In line (1) in main() the top-stories URL is fetched. Because http.get returns an ?Response, we need an error block to handle possible get errors. Resp is a Response struct value, and its text property contains the page’s html string. This is decoded in line (2) into an array of ints.

Only the first 10 stories are retained. In line (3) these are fetched by starting 4 coroutines with go. Line (4) assures that the program doesn’t close until all coroutines are finished.

In (5) the update to the story’s id cursus is locked: if this were not done, perhaps one story would be fetched multiple times, or a story with a certain id would not be fetched at all. The specific story is then fetched, decode into a Story struct value, and its title and URL are printed out.

## 13.8 V and databases

### A - sqlite module

First download sqlite from <https://sqlite.org/download.html>

Download the source code sqlite-amalgamation (contains the header files) and the precompiled binary for your platform.

On Windows:

Compilation error: fatal error: sqlite3.h: No such file or directory  #include "sqlite3.h"

?? Collect all the sqlite files in a folder with that name and put it in C:\v\thirdparty\sqlite : doesn’t work

?? Include e:\sqlite in PATH variable: doesn’t work

On Linux:

sudo apt-get install libsqlite3-dev

Example 1: Working with an in-memory sqlite database: *sqlite.v*

import sqlite

fn main() {

  db := sqlite.connect(':memory:')

  db.exec("create table users (id integer primary key, name text default '');")

  db.exec("insert into users (name) values ('Sam')")

  db.exec("insert into users (name) values ('Peter')")

  db.exec("insert into users (name) values ('Kate')")

  nr\_users := db.q\_int('select count(\*) from users')

  println('nr users = $nr\_users')

  name := db.q\_string('select name from users where id = 1')

  assert name == 'Sam'

  users := db.exec('select \* from users')

  for row in users {

    println(row.vals)

  }

}

?? concrete examples of B,C?D

### B – mysql module

### C – postgress module

### D – ORM module

## 13.9 Graphical modules.

Of special importance for the success of V are the graphical libraries, which make it easy to make apps with native UI capabilities: you no longer need to embed a browser to develop cross-platform apps quickly.

ui: *native cross platform GUI drawing library* (uses Cocoa on macOSX, winAPI/GDI+ on Windows, GTK+ on Linux); this uses native controls, for example, for a TextBox: NSTextView on macOS, edit HWND on Windows.

?? Coming: a Delphi-like visual editor for building native GUI apps.

For more complex 2D/3D applications:

gg: a small hardware accelerated graphics library that takes care of drawing primitives in modern OpenGL: writing shaders, compile them, initialize vertex buffer objects, working with text, etc.

It has built in camera logic.

DirectX, Metal, and Vulkan are going to be supported in the future (2019?), so you will be able to write cross platform graphical applications without worrying about implementation details.

It will allow loading and rendering complex 3D objects. By the end of this summer I’m hoping to implement skeletal animation as well. So it’ll be super easy to develop 3D games without any engines or complex IDEs.

*vid* uses gg for rendering

gl: an OpenGL wrapper

gx: Constants and helpers for drawing; used in time\_table.v / tetris.v

* Tetris works on Linux and Windows
* The Color struct is defined in the standard-library module gx, so by importing gx, you can write:

const (

red = gx.Color{r: 255, g: 0, b: 0}

blue = gx.rgb(0, 0, 255)

)

glfw (??): uses the OpenGL libraries

Right now this is desktop only, but mobile support is planned.

Examples:

1) see hot\_reload/bounce.v

2) see hot\_reload/graph.v

3) *tetris/tetris.v*: works on Windows and Linux

To compile on Windows: see document: How to compile tetris

?? See also: C:\v\vlib\ui\examples\users\_gui.v (Nov 20: not present)

See also: code\_examples\3D shooter (size +- 100 Kb): (*3dshooter.v + v2*)

## 13.10 vweb

<https://github.com/vlang/v/tree/master/vlib/vweb>

See module vweb (= GitHub source)

See examples/vweb in distribution

This modules uses the other vlib modules os, net, http and net.urllib.

It provides a MVC like web framework, like Rails or ASP.NET MVC, with an emphasis on simplicity.

It works with HTML templates, and it has a built-in V template language, which uses @ to indicate code. For example *index.html*:

@for post in posts

<div class=post>

<a class=topic href="@post.url">@post.title</a>

<img class=comment-img>

<span class=nr-comments>@post.nr\_comments</span>

<span class=time>@post.time</span>

</div>

@end

To this corresponds a method *index()* in a V source file, which invokes as its last line:

$vweb.html()

This compiles an HTML template into V during compilation, and embeds the resulting code in the current action. That means that the template automatically has access to that action's entire environment.

Deploying vweb apps:

Everything, including HTML templates, is **contained** in one binary file. That's all you need to deploy.

A simple example can be found at: example\_vweb

On Windows (Nov 29): it compiles and runs, an http-server responds on <http://localhost:8082/>

*Running vweb app on http://localhost:8082*

However, no text is shown in browser window.

When the browser client quits, the program stops with exit code 0

On Linux: same, compiles and runs, but browser shows 404 Not found, problem loading page

Other web frameworks:

**Vex**: <https://github.com/nedpals/vex>

inspired by Express and Sinatra

?? try out