The Hare programming language

A new systems programming language

Drew DeVault
SourceHut

December 27, 2022

Hare at a glance

Hare's design principles:

- Trust the programmer.
- Provide tools the programmer may use when they don't trust themselves.
- Prefer explicit behavior over implicit behavior.
- A good program must be both correct and simple.

Hare prioritizes simplicity, transparency, and robustness.

Hare at a glance

In practice:

- Simplifies C without a reduction in utility
- Static type system, C ABI superset
- Manual memory management
- No runtime
- Standardized
- Does not link to libc by default

Hare at a glance

Target use-cases: high-performance, low-level tasks

- Operating systems, drivers
- System tools, daemons
- Networking software
- Compilers and toolchains

Hare at a glance

```
use crypto::sha256;
use encoding::hex;
use fmt;
use hash;
use io;
use os;

export fn main() void = {
        const hash = sha256::sha256();
        const file = os::open("main.ha")!;
        io::copy(&hash, file)!;

        let sum: [sha256::SIZE]u8 = [0...];
        hash::sum(&hash, sum);
        hex::encode(os::stdout, sum)!;
        fmt::println()!;
};
```

The Hare toolchain

- harec: 17,000 lines of POSIX C11
- qbe: 11,000 lines of C89
- as + ld (binutils/llvm)

Hare is trivial to bootstrap. Let's do it live!

Arrays

Bounds checked:

```
let x: [32]int = [0...];
x[0];
```

Unchecked:

```
let x: [32]int = [0...];
let y: *[*]int = &x;
y[42]; // segfault!
```

Slices

```
let x: []int = [];
defer free(x);
append(x, 10);
append(x, 20);
append(x, 30);

assert(len(x) == 3);

delete(x[..2]);
insert(x[0], 42);
```

Static slices

```
let buf: [os::BUFSIZ]u8 = [0...];
let buf = buf[..0];
static append(x, 10);
static append(x, 20);
static append(x, 30);

assert(len(x) == 3);

static delete(x[..2]);
static insert(x[0], 42);
```

Strings

- UTF-8
- Few language features, too error-prone
- Batteries included in the stdlib

Tagged unions

```
let x: (int | uint | void) = 42u;
assert(x is uint);
assert(x as uint == 42);

match (x) {
case let x: uint =>
        fmt::printfln("x (uint): {}", x)!;
case let x: int =>
        fmt::printfln("x (int): {}", x)!;
case let void =>
        fmt::printfln("x (void)")!;
};
```

Tagged unions

```
let x: (int | uint | void) = 42u;
let ptr = &x: *struct {
        tag: uint,
        union {
                ival: int,
                uval: uint,
        },
};
// Tags are deterministic and derived from a hash of the type:
// uint:  7765258
// int:   158763829
// void:  4269177316
```

Error handling

Good news: we fixed C's crappy error handling!

```
type invalid = !void;
type syntaxerr = !(str, uint, uint);
type error = !(invalid | syntaxerr | io::error);
```

Error handling

```
fn canfail(...) (error | size) = {
        match (io::read(...)) {    // Explicit error handling
        case io::EOF =>
                abort("unexpected EOF");
        case let err: io::error =>
                fmt::fatal("I/O error: {}", io::strerror(err));
        case let z: size =>
                assert(z == len(buf), "underread");
        };

        let amt = io::write(...)!; // asserts on error
        amt += io::write(...)?;    // returns error to caller
        return amt;
};
```

Dependencies

- We have a module system!
- But no package manager
- Use your distro's package manager
- Choose your dependencies conservatively
- stdlib gets you most of the way there

Safety features?

Yep:

- Bounds checked arrays & slices
- Conservative optimizer
- Little to no undefined behavior
- Mandatory error handling
- No NUL terminators
- No magic integer casts
- No uninitialized variables
- You can't dereference a null pointer

Safety features?

But: double-free and use-after-free are possible.

- ASan catches these
- We might do a borrow checker

Other controversial matters

No first-class threads

- Language is fine but stdlib is not re-entrant
- I/O bound? Use event-driven I/O
- CPU bound? Use multiprocessing
- Aside: Helios supports threads

Generics?

- No generics or meta-programming
- Roll your own high-level data structures
- More like C in this respect than Hare's contemporaries

Statically linked only

- We might do PIC at some point (already done for aarch64)
- But we like static linking

The Hare standard library

- An interface to the host operating system
- Implementations of broadly useful algorithms
- Implementations of broadly useful formats and protocols
- Useful features to complement Hare language features
- Introspective meta-features for Hare-aware programs

The Hare standard library

- ascii
- bufio
- bytes
- crypto
- datetime
- dirs
- encoding
- endian
- errors
- fmt
- fnmatch
- format
- fs
- getopt
- glob
- hare
- hash
- io
- linux
- log
- math
- mime
- net
- os
- path
- regex
- rt
- shlex
- slices
- sort
- strconv
- strings
- strio
- temp
- time
- types
- unix
- uuid

The Hare standard library

This includes:

- Extensible I/O primitives
- Extensible filesystem abstraction
- Unix bits: fnmatch, glob, poll, etc
- A cryptography suite
- Comprehensive date/time support
- Regular expressions: POSIX ERE

Cryptography

High-level, easy to use API; plus low-level primitives, all implemented in Hare:

- AES (cipher modes: CBC, CTR, XTS)
- Argon2
- Blake2
- Blowfish
- ChaCha/XChaCha
- Curve25519
- ed25519
- HMAC
- Poly1305
- Salsa20/XSalsa20
- SHA-1, SHA-256, SHA-512
- RSA

TODO: Diffie-Hellman et al, DSA, X.509, TLS. Raising money for an audit.

Documentation

Comprehensive standard library documentation is available via haredoc in your terminal or online at harelang.org, along with many tutorials, guides, and the Hare specification.

Extended library collection

The Hare extended library collection provides various features which fall outside of the stdlib's scope but are still important to the ecosystem.

- Algorithms (e.g. hare-compress)
- Comprehensive operating system support (e.g. hare-linux)
- Databases (e.g. Redis, SQL)
- Important file formats (e.g. hare-xml)
- Graphics (e.g. image format decoders/encoders)

Selected projects written in Hare

- Helios: microkernel for x86_64, aarch64 (WIP), and riscv64 (soon)
- Himitsu: secret manager/password store
- btqd: bittorrent client, blocked on net::http
- scheduled: cron replacement
- hare-gl, glm, sdl2, etc: OpenGL bits
- hare-libui: GUIs, the easy way
- hare-wayland: GUIs, the hard way

What's next?

- A small number of language changes are pending
- Completion of the stdlib, especially cryptography
- Finalize the specification
- Self-hosted compiler
- New architectures and platforms
- Goal: simple, stable, and reliable: mostly unchanging

Help wanted

We need your help! Especially with these priorities:

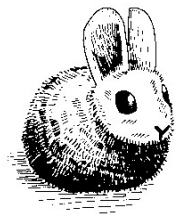
- New platforms
- Cryptography
- Maintainers for supported platforms
- Maintainers for the extended library

Or donate: `https://opencollective.com/hare`

Thank you, Hare contributors!

Adnan Maolood
Ajay Raghavan
Alexey Yerin
Andreas R
Andrew Chambers
Andri Yngvason
Antero Mejr
Armin Preiml
Armin Weigl
Ashish SHUKLA
Bor Grošelj Simić
Brian Callahan
Byron Torres
Carlos Une
Charlie Stanton
Christopher M. Riedl
Conrad Hoffmann
Drew DeVault
Edin Taric
Ember Sawady
Evan Vogel
Gabriel Schmotzer
George Rose
Haelwenn Monnier
Jason Lenz
Jean Dao
Joe Finney
Johann Freymuth
Jon Eskin
Jonathan Halmen
Jose Lombera
Josiah Frentsos
Karl Schultheisz
Kirill Primak
Kiëd Llaentenn
Lassi Pulkkinen
Lennart Jablonka
Louis Taylor
Miccah Castorina
Michael Forney
Mykyta Holubakha
Nihal Jere
Nikola
Nixon Enraght-Moony
Noah Altunian
Noah Loomans
Noah Pederson
Noam Preil
Nolan Prescott
Patrick Widmer
Pedro Lucas Porcellis
Pierre Curto
Pinghao Wu
Quentin Carbonneaux
Romain Reignier
Scott Little
Sebastian LaVine
Simon Ser
Steven Guikal
Sudipto Mallick
Thomas Jespersen
Tom Lebreux
Tom Regner
Umar Getagazov
Vincent Dagonneau
Vlad-Stefan Harbuz
Yasumasa Tada

The Hare programming langauge



`https://harelang.org`

Questions?