Gettin' **Ziggy** Wit It

By: Jake Lee, Sara Avila, & Seamus Gerner

Intro to Zig

- “General-purpose programming language”
  - Low-level Systems PL
  - Similar to C In Terms of What It’s Used For!
- Imperative + Procedural Programming Style
- Compiled Language, Not Interpreted
- Manual Memory Allocation
- Backend Development, OS, etc.

https://ziglang.org/

Background (4 W's)

Who?

- Andrew Kelley!

What?

- Created Zig!

When?

- Around 2016 - Present

Where?

- New York, NY

https://andrewkelley.me/post/intro-to-zig.html

Why was Zig created?

“I am nothing if not ambitious, and my goal is to create a new programming language that is more *pragmatic than C.* This is like trying to be more evil than the devil himself.” - Andrew Kelley

1. Pragmatic: Solves Your Problem!
2. Optimal: Fastest/Optimal Runtime!
3. Safe: Produces Safe/Reliable Outputs
4. Readable: Reading Over Writing!

https://andrewkelley.me/post/intro-to-zig.html

Type and Address Bindings

- Preference for Static Binding by Default
- Type Bindings at Compile-Time (comptime)
- Address Binding with Pointers (*T, *const T)
- Const -> Immutable
- Var -> Mutable
- **Optimal, Readable, Safe**

```
fn max(comptime T: type, a: T, b: T) T {
    return if (a > b) a else b;
}
fn gimmeTheBiggerFloat(a: f32, b: f32) f32 {
    return max(f32, a, b);
}
fn gimmeTheBiggerInteger(a: u64, b: u64) u64 {
    return max(u64, a, b);
}
```

```
pub fn main() void {
    var x: u32 = undefined;

    const tuple = .{ 1, 2, 3 };

    x, var y : u32, const z = tuple;

    print("x = {}, y = {}, z = {}\n", .{x, y, z});
```

https://ziglang.org/documentation/master/#comptime

Datatypes

- Full list of Primitive Types: ziglang.org/documentation/master/#Primitive-Types
- Focus on absolute precision, manual memory control, and compile-time evaluation
- Familiar Types: Boolean, Void, Null, Undefined
- Changes:
  - Integers are defined by their bit-width and range
    - Ex. i8 v.s. U8
  - Floats are chosen based on bit-width (no Double)
  - No Dedicated String Type
    - Instead, String Literals: constant single-item pointers to null-terminated byte arrays
  - No Dedicated Char Data type
    - Defined with u8
- **Pragmatic, Optimal, Readable**

https://ziglang.org/documentation/master/#Primitive-Types

Primitive Types

Primitive Types

Type	C Equivalent	Description
`i8`	`int8_t`	signed 8-bit integer
`u8`	`uint8_t`	unsigned 8-bit integer
`i16`	`int16_t`	signed 16-bit integer
`u16`	`uint16_t`	unsigned 16-bit integer
`i32`	`int32_t`	signed 32-bit integer
`u32`	`uint32_t`	unsigned 32-bit integer
`i64`	`int64_t`	signed 64-bit integer
`u64`	`uint64_t`	unsigned 64-bit integer
`i128`	`__int128`	signed 128-bit integer
`u128`	`unsigned __int128`	unsigned 128-bit integer

Familiar Control Statements

- **Pragmatic, Optimal, Safe**

While:

```
var i: usize = 0;
while (i < 5) : (i += 1) {
    // work here
}
```

If:

```
const x = 10;
if (x > 5) {
    // executes if true
} else {
    // executes if false
}
```

For:

```
const items = [_]i32{ 1, 2, 3, 4, 5 };

// Iterating over an array
for (items) |value| {
    std.debug.print("{d}\n", .{value});
}
```

Switch/Break/Continue:

```
test "switch continue" {
    sw: switch (@as(i32, 5)) {
        5 => continue :sw 4,

        // `continue` can occur multiple times within a single switch prong.
        2...4 => |v| {
            if (v > 3) {
                continue :sw 2;
            } else if (v == 3) {

                // `break` can target labeled loops.
                break :sw;
            }

            continue :sw 1;
        },

        1 => return,

        else => unreachable,
    }
}
```

https://ziglang.org/documentation/master/

Unique Control Statements

- **Pragmatic, Optimal, Safe**

Catch:

- used to handle an error locally by providing a fallback value or executing a block

```
const parseU64 = @import("error_union_parsing_u64.zig").parseU64;

fn doAThing(str: []u8) !void {
    const number = parseU64(str, 10) catch |err| return err;
    _ = number; // ...
}
```

Try:

- syntactic sugar for propagating an error up the call stack

```
const parseU64 = @import("error_union_parsing_u64.zig").parseU64;

fn doAThing(str: []u8) !void {
    const number = try parseU64(str, 10);
    _ = number; // ...
}
```

https://ziglang.org/documentation/master/

Defer:

- executes an expression unconditionally at scope exit, and in reverse order

defer_unwind.zig

```
const std = @import("std");
const print = std.debug.print;

pub fn main() void {
    print("\n", .{});

    defer {
        print("1 ", .{});
    }
    defer {
        print("2 ", .{});
    }
    if (false) {
        // defers are not run if th
        defer {
            print("3 ", .{});
        }
    }
}
```

Shell

```
$ zig build-exe defer_unwind.zig
$ ./defer_unwind

2 1
```

Errdefer:

- evaluates the deferred expression on block exit path if and only if the function returned with an error from the block

test_errdefer_capture.zig

```
const std = @import("std");

fn captureError(captured: *?anyerror) !void {
    errdefer |err| {
        captured.* = err;
    }
    return error.GeneralFailure;
}

test "errdefer capture" {
    var captured: ?anyerror = null;

    if (captureError(&captured)) unreachable {
        try std.testing.expectEqual(error.Gene
        try std.testing.expectEqual(error.Gene
    }
}
```

Shell

```
$ zig test test_errdefer_capture.zig
1/1 test_errdefer_capture.test.errdefer captu
All 1 tests passed.
```

Object-based and Inheritance

- Does NOT Directly Support OOP features (obj, classes, inheritance, etc.)
- However, there are “object-like” features:
- **Pragmatic, Optimal, Readable**

Structs:

a composite data type that groups related data into named fields; may include fields and methods

```
const Point = struct {
    x: f32,
    y: f32,
};

// Declare an instance of a struct.
const p: Point = .{
    .x = 0.12,
    .y = 0.34,
};

// Functions in the struct's namespace can be called with dot syntax.
const Vec3 = struct {
    x: f32,
    y: f32,
    z: f32,

    pub fn init(x: f32, y: f32, z: f32) Vec3 {
        return Vec3{
            .x = x,
            .y = y,
            .z = z,
        };
    }

    pub fn dot(self: Vec3, other: Vec3) f32 {
        return self.x * other.x + self.y * other.y + self.z * other.z;
    }
};
```

https://ziglang.org/documentation/master/

Tuples:

an anonymous struct that lacks field names, unlike arrays, tuple elements can have different types

```
const std = @import("std");

const MyTuple = struct {
    i32,
    u8,

    pub fn print(self: @This()) void {
        std.debug.print("{} {d}\n", .{self.@"0", self.@"1"});
    }
};

pub fn main() void {
    const t = MyTuple{ 10, 20 };
    t.print();
}
```

Data Structures - PT1

- Provides Several Built-In Data Structures
- Used to Build **Other** Data Structures:
- **Pragmatic, Readable, Optimal**

Arrays:

```
pub fn main() void {
    // Array of 5 integers, explicitly typed
    const a = [5]u8{ 1, 2, 3, 4, 5 };

    // Array with size inferred by compiler (_)
    const b = [_]u8{ 10, 20, 30 };

    // Array initialized with 'undefined' (must be mutable 'var')
    var c: [4]i32 = undefined;
    c[0] = 1;
}
```

Slice:

```
pub fn main() void {
    var array = [_]i32{ 1, 2, 3, 4, 5 };

    // Create a slice that points to elements 2, 3, 4
    const slice = array[1..4];

    // The slice length is known at runtime
    std.debug.print("Length: {}\n", .{slice.len}); // Prints: Length: 3
    std.debug.print("First: {}\n", .{slice[0]});   // Prints: First: 2
}
```

Enum:

```
// Declare an enum.
const Type = enum {
    ok,
    not_ok,
};

// Declare a specific enum field.
const c = Type.ok;
```

Union:

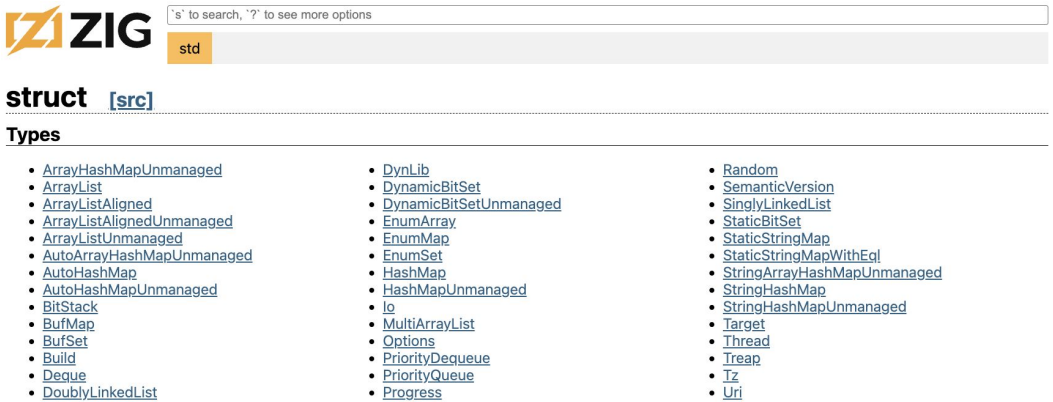
```
const Payload = union {
    int: i64,
    float: f64,
    boolean: bool,
};
test "simple union" {
    var payload = Payload{ .int = 1234 };
    payload.float = 12.34;
}
```

https://ziglang.org/documentation/master/

(Includes Structs/Tuples)

Data Structures - PT2

- List of Data Structures Included In the Zig Standard Library:



- Many Structures and Uses That We're Already Familiar With
- HOWEVER, Some New to Address Memory/Data Types
- **Pragmatic, Optimal, Readable**

https://ziglang.org/documentation/master/std/

Memory Management

- Manual Memory Management!
- Unlike C, DOES NOT Have a Default Allocator
- Functions Must Explicitly Recieve an Allocator
- All Four Allocators Serve a Different Purpose:

1. GeneralPurposeAllocator: Dev/Debug
2. ArenaAllcoator: Performance/Temp
3. FixedBufferAllocator: Embedded/No-Heap
4. PageAllocator:Large Allocations

Example of GeneralPurposeAllocator:

```zig
const std = @import("std");
const Allocator = std.mem.Allocator;
const expectEqualStrings = std.testing.expectEqualStrings;

test "using an allocator" {
    var buffer: [100]u8 = undefined;
    var fba = std.heap.FixedBufferAllocator.init(&buffer);
    const allocator = fba.allocator();
    const result = try concat(allocator, "foo", "bar");
    try expectEqualStrings("foobar", result);
}

fn concat(allocator: Allocator, a: []const u8, b: []const u8) ![]u8 {
    const result = try allocator.alloc(u8, a.len + b.len);
    @memcpy(result[0..a.len], a);
    @memcpy(result[a.len..], b);
    return result;
}
```

Shell

```
$ zig test test_allocator.zig
1/1 test_allocator.test.using an allocator...OK
All 1 tests passed.
```

https://ziglang.org/documentation/master/#Memory
https://paiml.com/blog/2025-02-18-zig-memory-management/

Sample Code

Real-World Implementation:

https://github.com/tigerbeetle/tigerbeetle/blob/main/src/queue.zig

```zig
const std = @import("std");
const assert = std.debug.assert;

const constants = @import("./constants.zig");

const QueueLink = extern struct {
    next: ?*QueueLink = null,
};

/// An intrusive first in/first out linked list.
/// The element type T must have a field called "link" of type QueueType(T).Link.
pub fn QueueType(comptime T: type) type {
    return struct {
        any: QueueAny,

        pub const Link = QueueLink;
        const Queue = @This();

        pub inline fn init(options: struct {
            name: ?[]const u8,
            verify_push: bool = true,
        }) Queue {
            return .{ .any = .{
                .name = options.name,
                .verify_push = options.verify_push,
            } };
        }

        pub inline fn push(self: *Queue, link: *T) void {
            self.any.push(&link.link);
        }

        pub inline fn pop(self: *Queue) ?*T {
            const link = self.any.pop() orelse return null;
            return @alignCast(@fieldParentPtr("link", link));
        }
```

```zig
// Non-generic implementation for smaller binary and faster compile times.
const QueueAny = struct {
    in: ?*QueueLink = null,
    out: ?*QueueLink = null,
    count: u64 = 0,

    // This should only be null if you're sure we'll never want to monitor `count`.
    name: ?[]const u8,

    // If the number of elements is large, the constants.verify check in push() can be too
    // expensive. Allow the user to gate it. Could also be a comptime param?
    verify_push: bool = true,

    pub fn push(self: *QueueAny, link: *QueueLink) void {
        if (constants.verify and self.verify_push) assert(!self.contains(link));

        assert(link.next == null);
        if (self.in) |in| {
            in.next = link;
            self.in = link;
        } else {
            assert(self.out == null);
            self.in = link;
            self.out = link;
        }
        self.count += 1;
    }

    pub fn pop(self: *QueueAny) ?*QueueLink {
        const result = self.out orelse return null;
        self.out = result.next;
        result.next = null;
        if (self.in == result) self.in = null;
        self.count -= 1;
        return result;
    }
```

Future

- Still a relatively new language!
- Bun: JS Runtime
- Tigerbeetle: Financial DB
  - Performance + Reliability
- Axiom, DNEG, Starknet, Syndica
  - Infrastructure + Backend
- Uber and Vercel
  - Build System and Tooling

https://github.com/rofrol/zig-companies-and-organizations

More on Zig

1. Zig Diary / Andrew Kelleys Website:https://andrewkelley.me/
2. Zig Website: https://ziglang.org/
3. Zig Documentation: https://ziglang.org/documentation/master/std/
4. Zig guide: https://zig.guide/
5. Live uses of Zig: https://github.com/rofrol/zig-companies-and-organizations
6. Zig github:https://github.com/ziglang
7. Youtube Guide:https://www.youtube.com/watch?v=UAS0f42_3uU

Questions?