DYALOG
Eastbourne 2026

# Codfns v6:
# Highlights and Roadmap



*Aaron W. Hsu – aaron@dyalog.com*

# What is Codfns?

- A compiler in APL for APL
- Expand the usefulness of APL
- An example of APL style/architecture
- Increase APL performance

# What is Codfns?

- “APL is the best systems language”
- “APL is the best high-level low-level programming language”

# What does that require?

- High Performance
- Control over behavior, resources
- Predictability
- Efficiency/Scalability
- Easy Integration

# Use Cases

- Large Data Science workloads
- Scientific Computing/HPC
- Performance critical functions/hotspots
- Embedded programming
- Non-permissive environments (Python, Web, Java)

# Use Cases

- Games
- Real-time applications
- “Systems Software”
  - Low-overhead Network servers
  - Operating System components

# What's New (v6)

- Lots of **Performance Improvements**
- Static (parametric) types for primitives
- Explicit storage handling
- Pool-based memory manager
- Standalone executables and I/O
- Parser Enhancements

# Performance

- Scalar-array optimizations
- Reduce unnecessary memory allocations
- Reduce primitive overheads
- Eliminate squeezing/promotions
- Better loop hinting for tight loops
- Static functions, no function allocation

# Performance

- Faster function calls
- Faster generation of derived functions
- Eliminate some unnecessary floating-point casts

# ISO APL Type Tower

- ISO APL has one type: the float
- All other types are sub-classes
- Interpreter optimizes representations

# New Type Taxonomy

- Introduce explicit numeric types:
  - Boolean
  - Exact (64-bit Integers)
  - Inexact (64-bit Floats)
  - Complex (Pair of Inexacts)
- Characters, etc.

# Vs. the Interpreter Taxonomy

- All Dyalog numeric values exist
- New Exact values exist that cannot be accurately represented inexactly
- (Like the interpreter) Inexact values may not convert precisely back and forth from exact values

# Why the new taxonomy?

- Previous: type promotion and squeezing
- No 64-bit integers!
- Excess promotion to floats
- Scanning is (more) expensive on GPUs

# Explicit Type Handling

- Primitives now define their output type solely on, at most, their input element type, not input element values
  - Except for ○ 😮
- General principle: “stay in your lane.”
  - Exact stays exact when possible
  - Inexact tends to make things inexact
- Idioms might keep things exact

# Benefits

- No more promotion or squeezing
- Tighter, higher performance primitives
- Better GPU performance
- Better fusion
- Less memory churn
- Sets the stage for static type inference

# The “Merge Map”

| | Bool | Int64 | Float | Cmpx | Char | Array |
|---|---|---|---|---|---|---|
| Bool | Bool | Int64 | Float | Cmpx | Array | Array |
| Int64 | Int64 | Int64 | Float | Cmpx | Array | Array |
| Float | Float | Float | Float | Cmpx | Array | Array |
| Cmpx | Cmpx | Cmpx | Cmpx | Cmpx | Array | Array |
| Char | Array | Array | Array | Array | Char | Array |
| Array | Array | Array | Array | Array | Array | Array |

# Explicit Casting and Storage

- Conversion to another type (Casting)
- GPU vs. Host Storage
- Array Squeezing
- Currently this is `⍬[<stg> <type>]A`

# Memory Management

- New memory manager
- Pool-based
  - Collections of free objects of a certain type
- Embedding friendly
- Almost immediate (de-)allocation

# Memory Management

- Reuses “hot memory”
- High amounts of reuse
- Low fragmentation
- Very easy to profiling
- No garbage collection
- Explicit function to reclaim memory

# Memory Management

- Provides control
- Retains convenience

# New Output Targets

- Linking a Dyalog Namespace
- Building a Shared Object/DLL
- Building a standalone EXE
- Running the program from the session

# Parsing API

- Waaay more language features parsed
- Will parse “bad” APL code without quitting halfway through
- Faster and uses less memory
- Significant features for flow analysis
- C.F. – Brandon Wilson

# Future Roadmap

- Release v6 this year
- Next Features:
  - Trains
  - Full “rich” assignments
  - Trad-fns + branching
  - Exception trapping

# Future Roadmap

- Other goodies:
  - Native Fenster-style graphics and audio API
  - WASM output

# Future Roadmap

- New Linking targets:
  - Python
  - Java
  - JavaScript
- Generate modules which wrap compiled Co-dfns programs/modules

# Conclusion

- New release by the end of the year
- Much better usability and performance
- Setting the stage for new integrations and features
- Static parser is **much** more capable now

# Thank you!

- New release by the end of the year
- Much better usability and performance
- Setting the stage for new integrations and features
- Static parser is **much** more capable now