# Adding Sequences to P0

Sujan Kandeepan, Razi Syed, Afzal Chishti

Date: April 2021

# Introduction

- Adding sequence data type to P0 language and compiler
- Support sequence operations:
  - Express sequences as literals and ranges
  - Sequence indexing and selecting subsequences
  - Concatenation of sequences
- Memory management due to dynamic size
- Similar to data types such as lists in Python

# Implementation

- Augment existing array type instead of defining new sequence type
- Extend language syntax to support new language features
  - Modify parser to accept new syntax and update grammar
  - Additional error handling with new syntax rules
- Array literals and subarrays as references to original array
  - Avoid duplicating array contents into new memory locations
  - New types/constructors easier code readability
- Overload existing operators adding support for arrays
  - Eg. ‘:=’ for populating entire arrays, ‘+’ for concatenation

# Testing the implementation - Validation

- 10 test cases of simple array literal assignment for integers
- 1 test case of simple array literal assignment for boolean
- 1 test case for testing indexing
- 2 test cases for multiple assignment
- 3 test cases for comparing arrays
- 3 test cases for testing subarrays
- 5 test cases for array concatenation

```
test1 = """
program p
    var a: [1..3] → integer
    a := [3, 7, 11]
    write(a[1])
    write(a[2])
    write(a[3])
"""
```

```
test4 = """
program p
    var a: [1..3] → integer
    var b: integer
    var c: boolean
    b, a, c := 42, [3, 7, 11], true
    write(a[1])
    write(a[2])
    write(a[3])
    write(b)
    if c then write(100)
"""
```

```
test14 = """
program p
    var a: [1..5] → integer
    var b: [1..5] → integer
    a := [1,2,3,4,5]
    b := [1,2,3,4,5]
    if a[1:4] = b[1:4] then write(100)
    if a[1:4] = b[2:5] then write(200)
"""
```

```
test15 = """
program p
    var a: [1..3] → integer
    var b: [1..3] → integer
    a := [3, 7, 11]
    b := [4, 8, 12]
    a := a + [5, 6]
    write(b[1])
    write(a[4])
"""
```

# Testing the implementation - Error checking

- 2 test cases for ensuring array literal fails when storing mixed types
- 12 test cases for testing out of bounds
- 1 test case for non-increasing ranges
- 1 test case for non-integer ranges
- 2 test cases to ensure only concatenation for two arrays of same type works
- 1 test case for concatenation

```
# Array literal with mixed types
test5 = """
program p
    var a: [1..3] → integer
    a := [3, true, 11]
"""
```

```
test7 = """
program p
    var a: [1..3] → integer
    a := [true..8]
"""
```

```
test8 = """
program p
    var a: [1..3] → integer
    a := [6..5]
"""
```

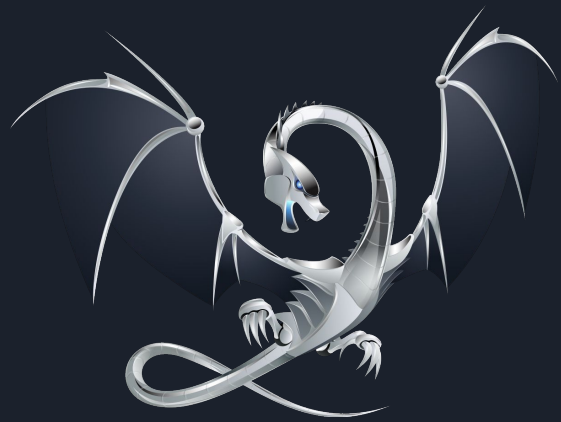
```
program p
    var a: [1..3] → integer
    var b: [1..3] → integer
    a := [3, 7, 11]
    b := [4, 8, 12]
    a := a + 5
    write(b[1])
    write(a[4])
"""
```

# Statistics

- 1252 number of lines
  - 179 - P0
  - 63 - CGwat
  - 2 - CGast
  - 343 - TestLiterals
  - 423 - TestIndex
  - 228 - TestConcat
  - 14 - ST
- 43 number of tests
  - 25 - Validation
  - 18 - Error checking

# Development

- Whether to add to array data type or create new
  - To prevent reduplication of work
  - Array already provided many features that would be expected of new data type
- Memory allocation
  - Looked into compilers such as Emscripten, Clang
    - Ton of overhead in these approaches
  - Wasted memory and shifting memory was out of context for this approach
  - Utilized memory.copy

# Conclusion

- Takeaways
  - Memory management
  - Insightful to work with
- Next steps
  - Array literals
    - Allow decreasing range (eg. s:=[5..1])
  - Subarrays
    - Allow negative indexing (eg. s[2: -2])
    - Allow one of the values for indices to be null (eg. s[:7])
  - Memory allocation
    - Shift bytes and memory address over

# References


1. Rossberg, A. (n.d.). WebAssembly Specification. Retrieved March 10, 2021, from https://webassembly.github.io/spec/core/
2. WebAssembly. (n.d.). Retrieved March 10, 2021, from https://developer.mozilla.org/en-US/docs/WebAssembly
3. Design and History FAQ. (n.d.). Retrieved March 10, 2021, from https://docs.python.org/3/faq/design.html#how-are-lists-implemented-in-cpython
4. Ahn, H. (2021, March 31). WebAssembly/bulk-memory-operations. Retrieved April 14, 2021, from https://github.com/WebAssembly/bulk-memory-operations/blob/master/proposals/bulk-memory-operations/Overview.md
5. Conrad Watt, Andreas Rossberg, and Jean Pichon-Pharabod. 2019. Weakening WebAssembly. Proc. ACM Program. Lang. 3, OOPSLA, Article 133 (October 2019), 28 pages. https://doi.org/10.1145/3360559
6. Building to webassembly. (n.d.). Retrieved April 12, 2021, from https://emscripten.org/docs/compiling/WebAssembly.html