XR/station Project
TOWER Language Rationale
Revision 1.0, March 23, 2023

TOWER Language Rationale

1. Introduction

This document describes the rationale for the new Tower language, which will replace Dragonfruit as the systems implementation language for the XR/station project.

Two significant bodies of code are currently written entirely in Dragonfruit, namely AISIX and MINTIA. The former is no longer maintained and will not be rewritten in Tower. The latter will be rewritten, one-to-one, in its entirety. It is anticipated that resolving bugs in the new Tower compiler will consume the largest amount of time during the project to rewrite MINTIA, so vigilance is extremely important during the writing of the new compiler.

2. Why a new language? Why rewrite MINTIA?

Dragonfruit was a useful and productive systems implementation language for 5 years, and there are over 150,000 lines of code written in it. Why throw it away?

The primary issue with Dragonfruit is that, through an unusual combination of C-like structures and reverse polish notation (RPN) expressions, it possesses a very opaque syntax that leads to nigh-inaccessible codebases. It is not quite a write-once, read-never language, but it could be called a write-once, read-never-by-anybody-but-you language.

Additionally, Dragonfruit has a complete lack of types, which has led to the construction of codebases that are heavily dependent on 32-bits, explicitly specifying 4 bytes whenever space for a pointer is needed, among other significant issues. Dragonfruit is also a stack-based language, which, while beneficial early in the project due to its easiness to write a trivial compiler for, turns out to be somewhat more difficult to write a vaguely optimizing compiler for.

Dragonfruit's multiple return is convenient, but is difficult to mesh well with traditional languages whose functions have a single return value and use passed pointers for extra return values.

Dragonfruit is very aesthetically unpleasing.

Among other issues.

The combination of these issues makes Dragonfruit the greatest problem of MINTIA, and it will likely be worth it in the long term to bite the bullet and rewrite it in a new language now while that's still somewhat feasible.

3. Characteristics of Tower

Tower will be designed to be a reasonable systems implementation language, with a primary goal of making it easy and simple to write clear and uncluttered code. Its syntax will be heavily inspired by Pascal, but compatibility with any variant of real Pascal is a non-goal. “Fancy” features and frills should be actively avoided in the design of Tower.

Compile-time language faculties will be present and will, at a minimum, take the form of:

- Conditional compilation (ifdef, ifndef, etc)
    - Very important for various portability issues.
- Include statements
    - With support for a “pragma once” type feature whereby subsequent includes of a file will be ignored. This should probably be the default so as to unclutter header files with include guards and make it a “don’t have to think about it” thing. This has to be considered deeply because header files under Dragonfruit got away with a lot due to typeless-ness, which won’t be possible when transliterated to Tower. Circular pointer fields may exist and Tower will need to either have a forward declaration feature, or a multi-pass parser to deal with this. [or back-patching!]
- Function macros
    - Will be useful for things like spinlock-guarded critical sections that are conditionally compiled to just disable interrupts on a uniprocessor build.
- Constant macros
    - Will be able to appear anywhere in the source text that a complete token could be, and will be substituted for the constant value. Will support rudimentary arithmetic operations. Function macros may be implemented as a special case of these (a constant macro followed by an open parenthesis).

These compile-time features may be implemented by any combination of preprocessor and compiler action.

Tower will be a portable language, both to 32-bit architectures other than its native XR/17032, and also to architectures with a greater (or lesser) bit width. Static type checking will be present, and a type inferring declaration operator will be present (probably as a composition of : and =).

Like Dragonfruit, a multiple-compilation paradigm will be fully supported. Unlike Dragonfruit, Tower will not possess an explicit multiple-return syntax, however you will be able to pass extra return values through pointers passed into the function, and the **OUT** keyword will cause the compiler to perform extra checks on the sanity of the usage of these pointers. The **IN** and **OUT** keywords will be able to be combined if desired.

TOWER Language Rationale

Tower will support the definition of new types of the following types:
- Structs
- Unions
- Function pointers (similar syntax to dragonfruit's fnptr)

Tower will support an inferred type declaration operator, **:=** , which will reduce the clutter of explicit type declarations. The declared variable will take the type of whatever is on the right side of the assignment.

Roadmap

- Initial bootstrap compiler w/ C backend.
- Self-hosting compiler w/ C backend, XR17032 backend, and fox32 backend.
    - Bootstrap compiler will no longer be maintained.
    - Compiler will be distributed alongside pre-"compiled" C sources so that an initial host cross-compiler can be built.