version 33⅓

Language Design Project

May 9, 2025

Aidan Carr

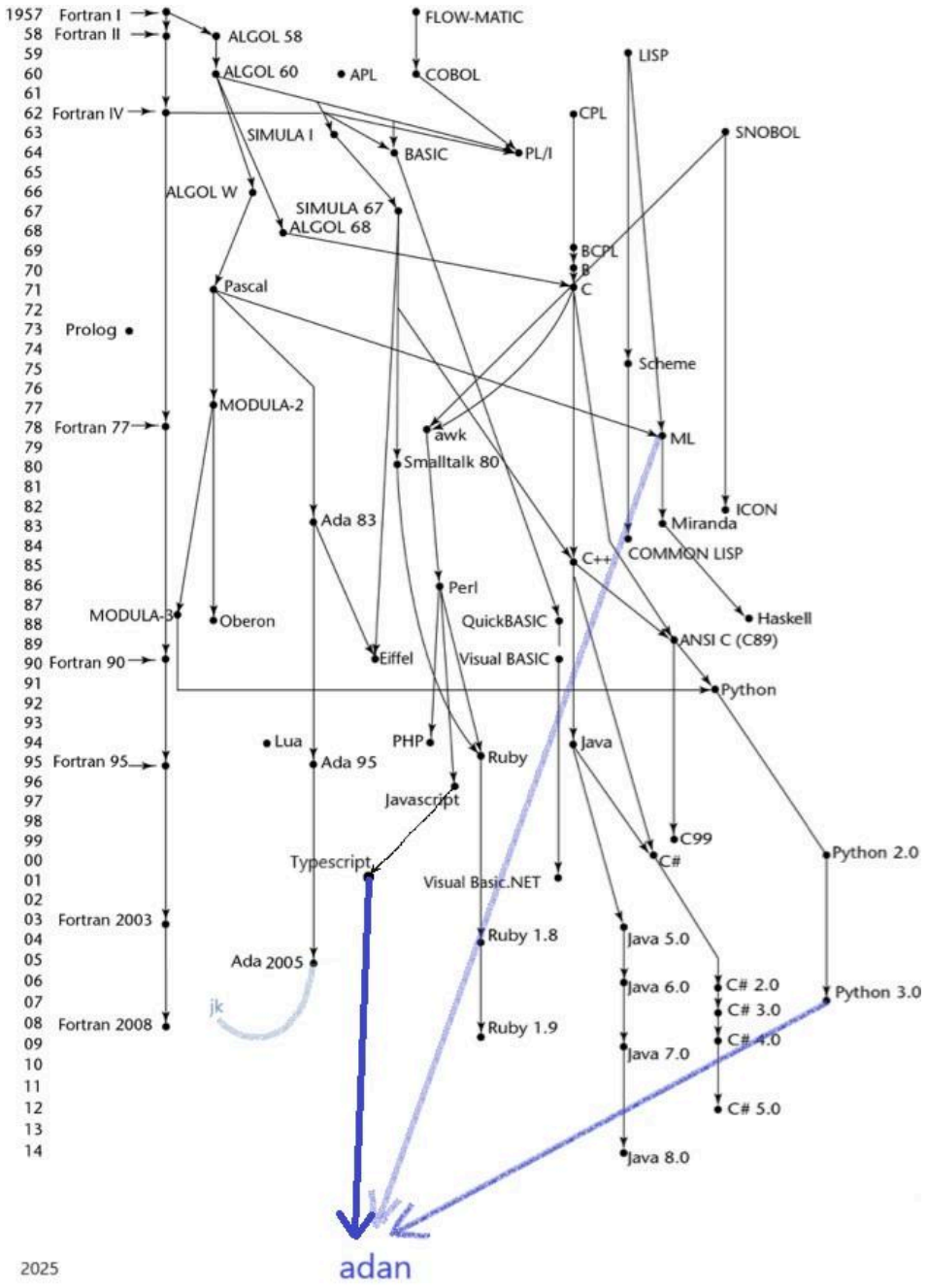
1 Introduction

The name of this language is a play on words of the programming language Ada (even though it is not related to Ada at all). It's close to my name, so I added an 'n' to the end of it. And the tagline `ada =+ n` is a reference to my favorite rule of the language, which hopefully defines itself.

The basis of this language is TypeScript, a more stable version of JavaScript. This language should be a more stable version of TypeScript, with some influences of Python and ML.

- The language is designed as an introductory coding language to math students, one that mixes all the best features of a few languages, while keeping syntax simple and easy to remember.
- Additional math instructions are included inside the standard library, while keeping numbers as only one data type, no integers, longs, floats, or reals. Behind the scenes it is stored as an improper fraction.
- The language has a general emphasis on consistency in declaring variables, constants, functions, and classes.
- String and array indexing functionality take ideas from Python.
- Character methods are inspired by ML
- Public and private methods are reserved for class methods while functions are just functions.

1.1 Genealogy



1.2 Hello World

```
1  function main()|void = {
2      writeln("Hello World");
3  }
```

1.3 Program Structure

Key organizational concepts include:

- The `main()` function is required and is just like another function. It is automatically called once. All code must be inside a function or class.
- Functions are defined only using `function` and are automatically static
- Classes are defined with the `class` keyword and their constructors with `new()`
- Class methods are called in a similar way, but they must be inside the class and are automatically non-static

```
1  namespace adan = {
2
3      ~ main function ~
4      function main()|void = {
5
6          ~ create a student ~
7          var student1|Student = Student("Christina", 2944, 3.5, true);
8          student1.writeStudent();
9      }
10
11     ~ class ~
12     class Student()|void = {
13
14         ~ constructor is called Student() ~
15         function new(nInput|String, iInput|number, gInput|number,
16                      eInput|boolean)|Student = {
17             att name|String = nInput;
18             att id|number = iInput;
19             att gpa|number = gInput;
20             att isEnrolled|boolean = eInput;
21         }
22
23         ~ print the student's info ~
24         function writeStudent()|void = {
25             writeln("Name: " + name);
26             writeln("ID: " + id);
27             writeln("GPA: " + gpa);
28             var status|String = isEnrolled ? "Is enrolled" : "Graduated";
29             writeln(status);
30         }
31     }
32 }
```

The namespace is declared as `namespace adan = {}` and inside there is a main function. A function named `main()` is automatically called after everything has been defined. There is also a class with a constructor, attributes, and a non-static method.

1.4 Types and Variables

Both value types and reference types still remain in adan. Variables/attributes that are a value type directly store data in their memory location. Reference types are objects, which directly store a pointer as an abstraction, which references another location in memory that stores the objects' attributes. Multiple references can be made to the same reference type, modifying it as it is stored deep in memory.

1.5 Visibility

There are no more visibility attributes. Everything is public and it is up to the programmer to include extra code to prevent data leaks.

1.6 Statements differing from TypeScript and Python

Statement	**Example**
Function Declaration	`function main()|void = {`
Variable Declaration	`    var PI|number = 3.1415;` `    var distance|number = 12;`
Is Equal Operator	`    var isZero|boolean = (PI ?= 0);`
And Or Statements	`    if (isZero OR (PI > 2 AND PI < distance)) {`

Print Statements	`        write("That is ");`
	`        writeln("interesting!");`
	`    }`

Percentage Operator	`    var half|number = 50%;`
Mod Operator	`    var remainder|number = 23 mod 5;`
Factorial Operator	`    var possibilities|number = 7!;`
Increment Operator	`    distance =+ 5;`
Decrement Operator	`    distance =- 5;`
Power Operator	`    var squared|number = distance ^ 2;`
Math Class Operators	`    writeln(#root(49));` `    writeln(#ceil(PI));`

Statement	Example
Class Declaration	} class House()|void = {
Constructor	new(add|String, own|String)|House = {

Attribute Declaration	att address|String = add;
	att owner|String = own;
	}

Method Declaration	function printZipCode()|void = {
Substring Operations	writeln(address[-5:]); }

String Comparison	function whatsUpDoc()|void = {
	if (owner[0:2] ?= "Dr") {
	writeln("Do you concur?");
	}
	}
	}

For loop	function oneThroughTen()|void = { for (var i|number = 1, i <= 10, i =+ 1) {

String concatenation	write(i + ", ");
	}
	}
	}

2 Lexical Structure

2.1 Programs

A program written in adan is stored as a standard text file with 0xA1DABEEF as a magic number (like Java's 0xCAFEBABE). The program is compiled in three steps

1. Lexical Analysis translates the user's text into predefined tokens.
2. The Parser converts the token stream into a valid Concrete Syntax Tree.
3. Semantic Analysis converts the CST into an Abstract Syntax Tree and checks scope and type of all variables, attributes, functions, and function parameters
4. Machine code is generated and executed on the computer using the AST.

2.2 Grammars

This specification presents the syntax of adan where it differs from TypeScript and occasionally steals from Python.

2.2.1 Lexical grammar tokens

Key changes include:

```
<Equal to Operator>   →   ?=
<Type Declaration>    →   “|”
<MORE Math Operators> →   ! | % | mod | #
<Type>                →   real
<Write Line>          →   writeln
<Write>               →   write
<Index Separator>     →   :
<Comment>             →   ~
<Increment>           →   =+
<Decrement>           →   =-
```

2.2.2 Syntactic parse grammar

Key changes include:

```
<AttDecl>  →   att <Decl>
<FunDecl>  →   function|class <FunDecl>
<Decl>     →   <Id> “|” <Type> = Statement;|<Block>
<FunDecl>  →   <Id> ( <ParamList> ) “|” <Type> = <Block>
<ForLoop>  →   for ( <VarDecl> , <Comparison>, <Statement> ) <Block>
<Inc>      →   <Id> =+ <NumberExpr> ;
<Dec>      →   <Id> =- <NumberExpr> ;
<StrIndex> →   <Id> | <String> “[” [<Number>] : [<Number>] “]”
```

2.3 Lexical Analysis

2.3.1 Comments

There is only one form of comment in adan, the multiline comment. The ~ symbol marks both the end and beginning of a comment. The lexer is in charge of determining start versus end tokens.

```
var isItSummer|boolean = ~ this is a comment ~ false;
```

2.4 Tokens

Tokens include:
identifier, keyword, punctuation, number-literal, character-literal, string-literal, boolean-literal.

2.4.1 Keywords different from TypeScript

Keywords that have been **added** to the TypeScript dictionary:
`real, writeln, write, att, mod, AND, OR, NAND, NOT, XOR, NOR, XNOR`

Keywords that have been **removed** from the TypeScript dictionary:
`public, private, const, static, console.log, .substring, Math`

3 Type System

The type system in adan is both strict and lenient. All variables, attributes, functions, and classes must have a variable type on declaration, which cannot change.

3.1 Type Rules

Assignment and Comparisons

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T \quad T \text{ is a primitive type}}{S \vdash e_1 \texttt{ = } e_2 : T}$$

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T \quad T \text{ is a primitive type}}{S \vdash e_1 \texttt{ != } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T \quad T \text{ is a primitive type}}{S \vdash e_1 \texttt{ < } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T \quad T \text{ is a primitive type}}{S \vdash e_1 \texttt{ ?= } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T \quad T \text{ is a primitive type}}{S \vdash e_1 \texttt{ > } e_2 : \text{boolean}}$$

Number Operations and String Concatenation

$$\frac{S \vdash e_1 : \text{number} \quad S \vdash e_2 : \text{number}}{S \vdash e_1 \texttt{ =+ } e_2 : \text{number}}$$

$$\frac{S \vdash e_1 : \text{number} \quad S \vdash e_2 : \text{number}}{S \vdash e_1 \texttt{ =- } e_2 : \text{number}}$$

$$\frac{S \vdash e_1 : \text{number} \quad S \vdash e_2 : \text{number}}{S \vdash e_1 \texttt{ mod } e_2 : \text{number}}$$

$$\frac{S \vdash e_1 : \text{number} \quad S \vdash e_2 : \text{number}}{S \vdash e_1 \texttt{ + } e_2 : \text{number}}$$

$$\frac{S \vdash e_1 : \text{number} \quad S \vdash e_2 : \text{number}}{S \vdash e_1 \texttt{ - } e_2 : \text{number}}$$

$$\frac{S \vdash e_1 : \text{string} \quad S \vdash e_2 : \text{string}}{S \vdash e_1 \texttt{ + } e_2 : \text{string}}$$

Logic Gates

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ OR } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ NAND } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ NOR } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ AND } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ XOR } e_2 : \text{boolean}}$$

$$\frac{S \vdash e_1 : \text{boolean} \quad S \vdash e_2 : \text{boolean}}{S \vdash e_1 \texttt{ XNOR } e_2 : \text{boolean}}$$

3.2 Value Types

Types MUST be declared with every variable and function. Certain data types have been removed to keep the language simple. All numbers are one data type: number. Behind the covers, this is stored as a numerator and denominator and keeps track of significant figures.

Additionally, there are no more constants; naming conventions are still recommended for constants. Attributes are a version of variables that replace the `this` keyword, but are variables linked to the object/class instance.

3.3 Reference Types

Reference types include programmer-defined objects and adan-defined objects like Strings and Arrays. There are no longer multiple string/String types from TypeScript, just the one String type.

4 Example Programs

4.1 Caesar Cipher Encrypt

```
function encrypt(text|String, value|number)|String = {

    ~ edit value ~
    value = value mod 26;
    var caesarString|String = "";

    ~ encrypt each letter ~
    for (var i|number = 0, i < text.length, i =+ 1) {
        var character|char = text.toUpperCase()[i:i+1];

        ~ letter ~
        if ('A' <= character AND character <= 'Z') {
            character =+ value;

            ~ loop back after Z ~
            if (character > 'Z') {
                character =- 26;
            }
        }

        ~ push ~
        caesarString =+ character;
    }
    return caesarString;
}
```

4.2 Caesar Cipher Decrypt

```
function decrypt(text|String, value|number)|String = {
    return encrypt(text, 26-value);
}
```

4.3 Factorial

```
function factorial(x|number)|number = {
    ~ return x! is another option in this language ~
    if (x ?= 0) {
        return 1;
    }
    else {
        return x * factorial(x-1);
    }
}
```

4.4 Selection Sort

```
function selectionSort(items|number[])|void = {
    var minPosition|number;

    for (var i|number = 0, i <= items.length - 2, i =+ 1) {
        minPosition = 1;

        for (var j|number = i+1, j <= items.length - 1, j =+ 1) {
            ~ compare ~
            if (items[j] < items[minPosition]) {
                minPosition = j;
            }
        }
        ~ swap items ~
        var temp|number = items[i];
        items[i] = items[minPosition];
        items[minPosition] = temp;
    }
}
```

4.5 Card Shuffling

```
namespace cards = {
  function main()|void = {

    var deck|String[] = createCards();
    shuffle(deck);

    writeln("Is this your card?");
    writeln(deck[0]);
  }

  function createCards()|void = {
    var suits|String[] = ["hearts", "spades", "diamonds", "clubs"];
    var values|String[] = ["ace", "one", "two", "three", "four", "five",
          "six", "seven", "eight", "nine", "ten", "jack", "queen", "king"];
    var cards|String[];

    ~ loop every suit ~
    for (var s|number=0, s < suits.length, s =+ 1) {

      ~ loop every value ~
      for (var v|number=0, v < values.length, v =+ 1) {

        ~ push the name of each card ~
        cards =+ values[v] + " of " + suits[s];
      }
    }
  }

  ~ pass by reference ~
  function shuffle(items|String[])|void = {
    for (var i|number = items.length-1, i > 0, i =- 1) {
      var randomIndex|number = #floor(#random() * (i + 1));

      ~ swap cards ~
      var temp|String = items[i];
      items[i] = items[randomIndex];
      items[randomIndex] = temp;
    }
  }
}
```

4.6 Stack

```
namespace myStack = {
  function main()|void = {

     var schoolWork|Stack = Stack();
     schoolWork.push("Math worksheet");
     schoolWork.push("Science quiz");
     schoolWork.push("Self portrait");

     ~ print tasks in reverse ~
     writeln(schoolWork.pop());
     writeln(schoolWork.pop());
     writeln(schoolWork.pop());
  }

  ~ objects stored in the stack ~
  class Node()|void = {
    function new(nInput|String)|Node = {
      att name|String = nInput;
      att next|Node = null;
    }
  }

  ~ stack class only knows the top and how to deal with it ~
  class Stack()|void = {
    function new()|Stack = {
      att top|Node = null;
    }

    ~ add to the top ~
    function push(strInput|String)|void = {
      var newNode|Node = Node(strInput);
      newNode.next = top;
      top = newNode;
    }

    ~ remove and return top ~
    function pop()|Node = {

      ~ if empty Stack ~
      if (top ?= null) {
        return "ERROR, EMPTY STACK";
      }
      else {
        var popping|String = top.name;
        top = top.next;
        return popping;
      }
    }
  }
}
```