Practical 1- Research Assignment

Hayley Dodkins u21528790

18 February 2025

1

Esoteric programming languages also known as esolangs are designed to be unconventional. They can be categorised into the following types:

- Objective-Driven Languages: These languages were designed to achieve an objective such as producing as few instructions as possible or languages to be used in code golf where the goal is to have the shortest source code to solve a given algorithm [14, 15].
- Unusable languages: Designed to be intentionally difficult to write. These languages often restrict the use of symbols. A well-known example would be a language called Brainfuck which consists of only 8 commands [15]. Concept-exploration languages introduce new programming concepts. Befunge arranges code in a two-dimensional grid allowing execution in multiple directions [15].
- Artistic languages: These languages are designed to resemble something other than traditional programming source code. For instance, Piet is a language designed to look like abstract art, while Shakespeare is a language that emulates the style of Shakespearean play [14].
- Joke languages: Created for humour. INTERCAL was a language designed that requires the programmer to use the keyword "please" an appropriate number of times, if not used enough the compiler deems the code impolite, if used too much the programmer is overly ingratiating [2].

2

2.1 Funges

Funges are difficult to compile. The compiler cannot perform a static analysis and determine the control flow of a program and, due to the self-modifying behaviour, it is hard for the compiler to make any traditional modifications [12,13]. Funges enable a way to explore alternative computation models different to the

classic Von Neumann architecture where there is linear instruction execution, Funges explore the idea of multi-dimensional computing and non-traditional execution flows [9].

2.2 Turing Tarpits

Turing tarpits are considered Turing complete, meaning that theoretically the language can be used to solve any computational problem given enough memory and time [5]. Turing tarpits are considered practically unusable because they are often difficult to write due to only using a limited number of symbols [15].

3

3.1 INTERCAL

Year of Initial Design: 1972
Designers: Donal R. Woods and James M Lyon

General Characteristics: The goal of INTERCAL was to be a compiler language with nothing in common with any other major language at the time. The full name of the language is "Compiler Language With No Pronounceable Acronym", which is abbreviated to "INTERCAL" [1].

Variable Types: INTERCAL has five types of variables [3]:

- The 16-bit unsigned integer (.) followed by a number
- The 32-bit unsigned integer (:) followed by a number
- The array of 16-bit unsigned integers (,) followed by a number
- The array of 32-bit unsigned integers (;) followed by a number
- Constants are 16-bit unsigned integers () followed by a number

Operators: INTERCAL has 5 operators, two binary operators and three unary operators [1].

- Interleave (¢): A binary operator that takes two 16-bit values and produces a 32-bit result by alternating the bits of the operands.
- Select( ): Extracts bits from one operand based on the second operand's 1-bits.
- The unary operators: (logical AND), V (logical OR), and V (logical XOR).

Syntax

- INTERCAL consists of a list of statements preceded by an optional line label followed by DO, PLEASE, or PLEASE DO [1].
- INTERCAL enforces a "politeness rule", where 1/3 to 1/5 of statements must include "PLEASE." If too few or too many of the statements are polite, the compiler will reject the program [1].
- The angle (¡) followed by a worm (-) is the equivalent of the assignment operator (=) in FORTRAN [1].
- The NEXT statement is used for subroutine calls and unconditional transfers where DO (label) NEXT and PLEASE DO (label) NEXT are the two forms of this statement [1].
- The statement PLEASE FORGET exp, causes the expression to be evaluated, and the specified number of entries to be removed from the stack [1].
- PLEASE GIVE UP is used to exit from a program [1].
- Input is taken with the statement DO WRITE IN x, where x would be a variable [1].
- INTERCAL does not have an equivalent to an IF statement or GO TO, instead DO ABSTAIN FROM and PLEASE ABSTAIN FROM is used to ignore all statements of specified types [1].

[1]

```
      DO ,1 <- #1
      DO .4 <- #0
      DO .5 <- #0
      DO COME FROM (30)
      DO WRITE IN ,1
      DO .1 <- ,1SUB#1
      DO (10) NEXT
      PLEASE GIVE UP
(20)  PLEASE RESUME '?.1$#256'~'#256$#256'
(10)  DO (20) NEXT
      DO FORGET #1
      DO .2 <- .4
      DO (1000) NEXT
      DO .4 <- .3~#255
      DO .3 <- !3~#15'$!3~#240'
      DO .3 <- !3~#15'$!3~#240'
      DO .2 <- !3~#15'$!3~#240'
      DO .1 <- .5
      DO (1010) NEXT
      DO .5 <- .2
      DO ,1SUB#1 <- .3
(30)  PLEASE READ OUT ,1
```

Figure 1: This example code was taken from [1].

3.2 Piet

Year of Initial Design: 2001
Designers: David Morgan-Mar
Piet is a visual esoteric programming language in which programs resemble abstract paintings inspired by Piet Mondrian [4]. The program execution follows a direction pointer that moves across coloured blocks (codels), interpreting colour transitions as commands [8].

Data handling: Piet is stack-based using only integer values [8].
Tokens:

- The smallest entity is a codel which consists of a single pixel or many pixels depending on the size of the image [7].
- A colour block is a continuous block of codels of the same colour [7].

Program traversal:

- An interpreter starts at the top left codel (this codel cannot be black). The interpreter moves from one colour block to another using the direction pointer and codel chooser to decide which colour block is next [7].
- The direction pointer can move in four directions, up, down, left or right. This pointer initially points right [7].
- The codel chooser can point left or right. At the start the codel chooser points to the left. The interpreter uses the codel chooser to choose which one of the neighbouring colour blocks along the border is the next one to move into [7].
- The edges of a program and black colour blocks cannot be penetrated and limit possible movement options [7].
- If the interpreter moves into a white colour block, it follows the current direction of the direction pointer until a colour block is reached or a black colour block or edge is found [7].

Colour-based Commands:

- When the interpreter moves from one colour block to the next one. It triggers the execution of commands [7].
- Commands are encoded in the amount of steps taken in hue and lightness of the colour transition [7].
- Depending on the command executed an integer is either pushed onto the stack or popped off the stack [7].

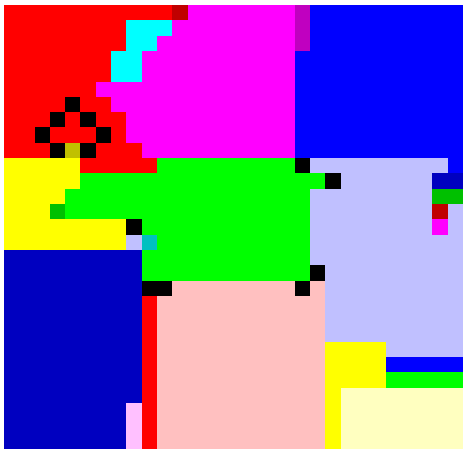

Figure 2: Taken from [8], the example Piet code above prints "Hello world!" and then exits.

4

Design by contract (Dbc) is a software design approach where contracts (interaction protocols) are used to assert how classes and methods should interact. Dbc is achieved through the following concepts [11]:

- **Preconditions:** Assertions that must be true when a function is called [6].
- **Postconditions:** Assertions that hold true once a function has been completed [6].
- **Invariants:** Classes or properties that should remain unchanged when there is interaction between participants [6].

Both Clojure and Ada have native support for Design by contract [10].

5

"Vibe programming" refers to a programming approach where developers rely heavily on AI coding tools to generate and debug code without deep understanding or analysis. When the AI tools fail, the developer results to random fixes until something works rather than systematically debugging the code.

- **No critical thinking:** The developer does not engage in problem-solving but instead depends on AI-generated solutions. This reduces their ability to evaluate, debug or optimize code independently. Without the necessary critical thinking skills, developers cannot fix issues which will lead to inefficiencies when AI tools are not available or produce suboptimal solutions.

- **Lack of innovation:** AI is trained on existing code, meaning AI generates standard and widely used solutions rather than optimized more advanced approaches. This could stifle innovation, as AI would favour well-known solutions over new languages, frameworks or methodologies making it difficult for emerging technologies to gain traction.

- **Reliance on AI Tools:** If developers rely entirely on AI tools for coding, any downtime, outage or limitation can render them unable to do their job. They would struggle to understand the AI-generated code or any error messages. This type of dependence creates a fragile workflow where productivity is tied to the availability of AI-tools rather than the programmer's skill set.

- **Lack of programming knowledge:** A developer who only codes through the use of AI assistance may never gain a deep understanding of programming languages and their features. Since AI-generated code is based on common patterns the developer will never learn how to use more advanced features. These developers would struggle to assess if a piece of code is truly good or just good enough as they lack the foundational knowledge.

References

[1] Louis Howell Eric Raymond Donald Woods, James Lyon. The intercal programming language revised reference manual. *n.d*, 1996.

[2] Gyau Elvis. Esoteric languages: What are they, and why you should be concerned? *Dev*, 2024.

[3] Esolangs. Intercal. *Esolangs*, n.d.

[4] Esolangs. Piet. *Esolangs*, n.d.

[5] Mark Harrison. What is turing complete? *StackOverflow*, 2020.

[6] kaushik. Design by contract it's relevance in game programming. *Medium*, 2020.

[7] Manfred Moosleitner. Piet an artistic programming language. *Seminar Report*, 2015.

[8] David Morgan-Mar. Piet. *DM's Esoteric Programming Languages*, n.d.

[9] Sebastian Morr. An introduction to brainfuck, intercal, befunge, malbolge, and shakespeare. *Blinry*, n.d.

[10] n.d. Design by contract. *Wikipedia*, n.d.

[11] Dave Nicolette. Design by contract: Part one. *Leading Agile*, n.d.

[12] Oak. Why is befunge considered hard to compile? *Stackoverflow*, 2016.

[13] Justin Ohms. Arcane programming: Befunge. *Medium*, 2023.

[14] Thomas Stuart. Hello, world! the best of esoteric languages. *The Quality Duck*, n.d.

[15] Gabriele Tomassetti. The fun (and madness) of esoteric programming languages. *Strumenta*, n.d.