MODULE: PROGRAMMING LANGUAGE CONCEPTS

SYSTEMATIC LITERATURE REVIEW

(COURSEWORK 1)

NAME: CHEE ZI HAN

STUDENT ID: 33354596

Student Declaration

I hereby declare that this coursework is written by me and is a result of my own work. I have not copied ideas/research and academic sources.

Contents

INTRODUCTION

A programming language consists of a set of rules and steps meant for a computer's understanding to execute a task (Goyal et al., 2020). Programming languages allow people such as developers to communicate with computers through machine, high level, and assembly language, and thus allowing for more innovation and advancements in the world of technology and daily life (Goyal et al., 2020).

The objectives of this systematic literature review are to cross examine existing literature in the following aspects:

1. Identify and understand fundamental programming language concepts.
2. Analyse and interpret trends in programming languages.
3. Explore and assess advancements in programming languages.
4. Evaluate impact of programming languages on industry practices, research and education.

The scope of the review involves the following language concepts:

1. Syntax
2. Semantics
3. Data types and structures
4. Control structures
5. Functions
6. Concurrent Programming

In addition, trends regarding how programming languages develop according to industry demand will be identified and explored, providing a deeper understanding on the impact of programming languages in the field of computer science through the decades.

LITERATURE SEARCH STRATEGY

The databases used are Google Scholar, ACM Digital Library, Science Direct, Citeseer library, and IEEExplore.

The search terms used were:

- Programming Language Concepts
- Abstract Data Types
- Control statements structured programming

- Functions in programming
- Evolution of programming
- History of programming languages
- Future of programming language
- Programming Languages in education

The most used and relevant database for this literature review was Google Scholar during searching, so only Google Scholar will be considered in this literature review.

The Boolean values used were ‘AND’ and ‘OR’, combining the search terms above.

INCLUSION AND EXCLUSION CRITERIA

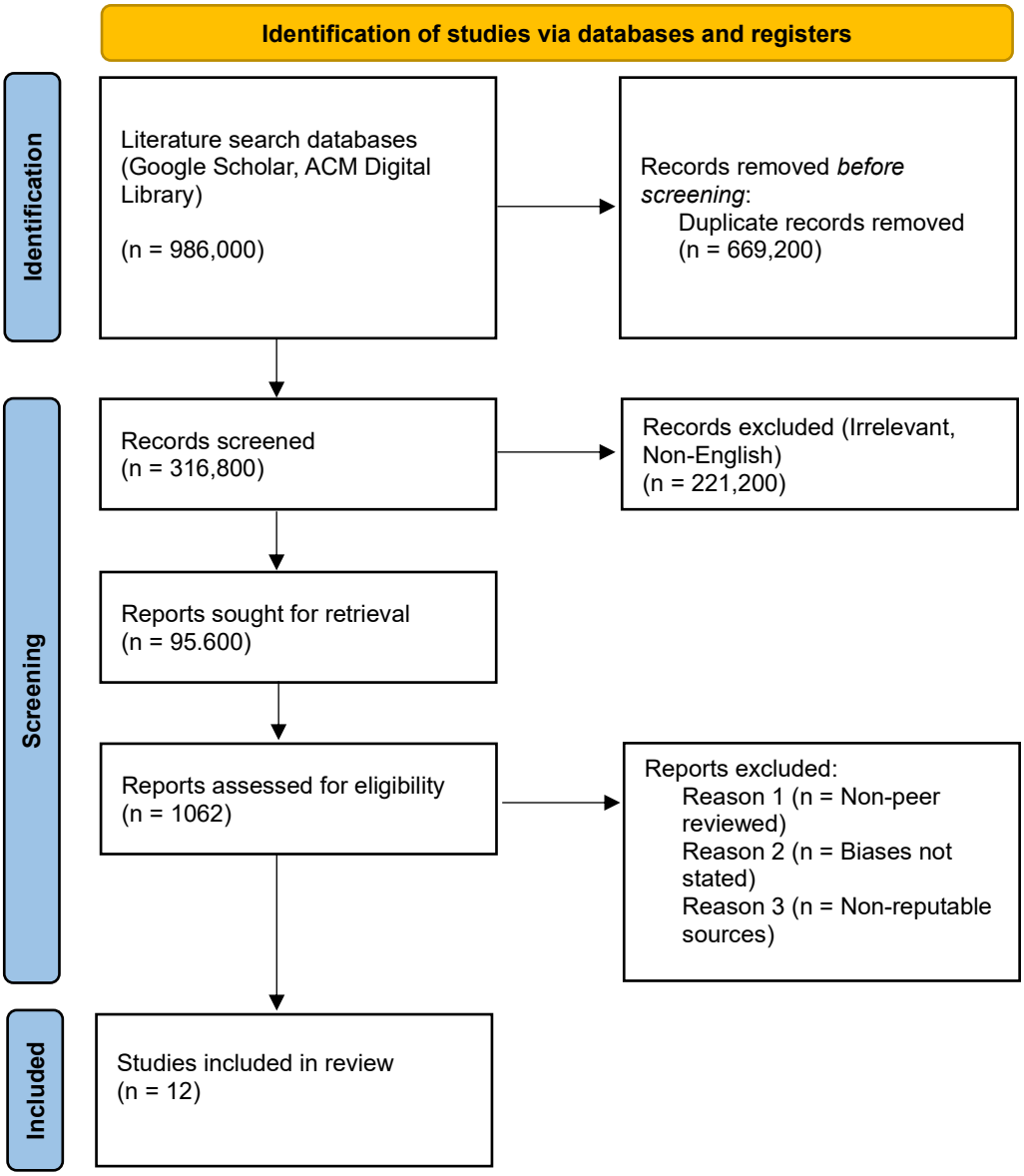
Inclusion criteria:

Material	• Peer-reviewed journal articles • Conference papers • Books
Relevant information	• Programming language concepts • Theories • Trends • Advancements
Language Used	• English

Exclusion criteria:

Material	• Blogs • Forums • Other non-peer-reviewed sources
Language Used	• Non-English

Below is a PRISMA Flow Diagram (*PRISMA 2020 Flow Diagram — PRISMA Statement*, n.d.) that illustrates the search and selection strategy:



DATA EXTRACTION

The process of extracting data from the chosen literature include choosing literature that only had information about programming language concepts such as:

- Syntax
- Semantics
- Data Types
- Control Structures
- Functions
- Trends
- Advancements and evolution of programming languages
- Programming languages in education

The information was chosen according to:

- Title
- Author
- Abstract

It is to be noted that the publication year was not taken into consideration. If the information was relevant and cross checked, it is used in the literature review.

QUALITY ASSESSMENT

The quality assessment were done in considered to the factors below:

- Related to programming language concepts
- Peer-reviewed journals
- Reputable conference proceedings
- Books authored by experts
- Papers that acknowledge own biases

SYNTHESIS AND ANALYSIS

The first concept is syntax, where a language is defined by expression that are validated by a fixed set of rules for that language (Ben-Ari, 1996). Syntax can be identified via two types, which are concrete syntaxes that includes blank spaces, brackets and parentheses, and abstract syntaxes which can be represented with trees and abstracted blank spaces and parentheses. [add syntax comparison of old and new languages / concept of syntax]

One concept closely related to syntax is semantics, or the context of an expression written in a programming language. A semantic can describe the content within the program's memory, and it provides the following instruction to be done.

Next are data types. This is a core concept for programming concepts, as they represent a value and a certain set of instructions attached to those values (Ben-Ari, 1996). Types include primitive data types built into programming languages, including int, char, bool, etc (Cardelli & Wegner, 1985). There are also abstract data types (ADTs), which are data specified using higher-order equations through constructors and operations (Jean-Pierre & Mitsuhiro, 1997), where examples include stacks used for data structures like lists or arrays containing data such as integers. With data types, comes static and dynamic typing. There is a distinct trend in programming language types, as early as the years of FORTRAN, where integers and floating-points were identified by the first letter, but ALGOL 60 introduced identifier declarations. PL/I then followed with typed arrays and pointers but provided weak compile-time type checking, to which Pascal attempted to solve this problem but introduced another problem of type equivalence. Then, object-oriented languages were introduced, such as Simula, Modula-2 and Ada, providing polymorphism in operations (Cardelli & Wegner, 1985).

Control structures are also a common concept in programming, where it allows for a defined order of execution of assignment statements. Well-structured control statements include choice and loop statements, which have if-statements and case-statements and for and while statements respectively. For if-statements, expression execution is determined by a Boolean value, and the implementation differs in programming languages (Ben-Ari, 1996).

Another concept in structured programming is the loop statement, where statements are executed repeatedly until it reaches a condition that can cause an exit or a break from the loop. It is challenging to program, being prone to errors due to incorrect boundaries and its inefficiency if declared wrongly. Loop statements include for, while and do while loops (Ben-Ari, 1996).

Another concept are subprograms. A subprogram is a chunk of written code that can be called when needed within the bounds of the program. It includes the declaration of the subprogram, such as its name, parameter list if any apply, and the type to be returned. Within it also exists local declarations which cannot be accessed outside of the subprogram, and a number of statements to be executed. A subprogram that returns a value is called a function, while a subprogram that does not return a value are procedures. Subprograms are useful when statements need to be executed repeatedly at different stages of the program, which in turn enhances readability, memory saving, and efficient testing (Ben-Ari, 1996). It is important that for a function, the result returned after statement execution is the same as the value declared to be returned. Because of this, some confusion may arise on the compiler's side if the declared return type and actual return type are different, thus implicit type conversion is introduced such as in C, compared to Ada where both types must be the same.

One other programming language concept is concurrency, or concurrent programming, where several things are being executed at the same time. This kind of program uses a concept called threading, where it uses threads to ensure that the processes are run in parallel (Hemmendinger, 2003). When a device has multiprocessors, parallel processors and distributed systems, concurrency will run by default. Operating systems that have time sharing support as well as input and output (I/O) devices make use of concurrency. An advantage of concurrent programming is the use for modular programming. Several processes will be executed at the same time, or in sync. This process of synchronisation involves mutual exclusion and condition synchronisation. This means the processes must take turns, which is ensuring one process fulfils a condition before its related next process is key to proper synchronisation.

DISCUSSION

Before delving into further discussion, an interesting train of thought regarding the terms of programming concepts by Strachey (2000) raises the question of the vagueness or 'matter-of-factly' way programming language concepts are named and referred to. This is because concept such as 'value' or 'set' already have semantics that exist within mathematics and in daily life so creating neutral concept names is certainly a potential field of future research.

As technology becomes more advanced and times change, naturally, programming language concepts must also adapt to the programming language it is designed for. It is also to be emphasized that programming languages themselves undergo transformation as 11 new languages are born every decade, as observed by Chowdhary (2020). The first computer was born from the help of Charles Babbage and Ada Lovelace (Linda Weiser, 1992), to which aided in the birth of programming languages. Following the discovery of assembly language, FORTRAN, COBOL, LISP and ALGOL were created, with FORTRAN being the first modern programming language by 1955, and C was released in 1969. Chowdhary (2020) also notes that as programming languages progressed and improved, some features were discarded, and some were continued via mutation of evolution theory. An example of this was C had the control feature from FORTRAN and not the goto statement. Programming languages such as AWK and Shell were left in favour of Perl and Python in terms of scripting languages.

Another potential area of research is in terms of language typing, which are static and dynamic typing. The argument is static typing makes code more verbose, unsafe, less reusable, and inexpressive compared to its counterpart, dynamic typing. Meijer and Drayton (2004) note that in modern times, some prefer dynamic typing as it allows for seamless integration of data intensive applications, however they emphasize that focusing on the strengths of both types may be more ideal as context is crucial. They note that no concrete study has been done on the possibility of a hybrid type.

There is also a noticeable change in the concept of control structures in how programming languages determine if a section of code should be executed based on a certain criterion. To recall, control structures like if-statements will execute a number of statements if criteria like a Boolean value of 'True' is met, otherwise another set of statements will be executed. As C does not have a Boolean type, it uses integer values in its place, where true equates to non-zero values and zero value as false (Ben-Ari, 1996). In comparison, later languages like Java and Python do have Boolean values. In contrast is the switch case statement, where the condition

is an exact value, otherwise the default statement is executed. Another difference is the usage of 'break', that can be seen in C programming, explicitly halting the switch case statement if an exact value has been met. Ben-Ari (1996) also notes that one disadvantage of the switch case is needing to know the values to be detected, otherwise the default statement will be executed, and run time checks can prove time consuming and inefficient should a desired output not be met.

For the concept of loops, there were also doubts surrounding the usage of utility goto in FORTRAN, and further research for higher level control structures are needed. In a paper by Ledgard and Marcotty, they state that programs using goto versus without can be more efficient and both provide the same clarity. On the other hand, a paper by Ishihata and Hikita propose a generalised loop exit statement which can better represent a postlude action for each exit to handle events. However, this proposal is not suitable to PASCAL as it cannot handle certain cases in exception handling (Yasamura, 1997).

There is also a trend in subprograms, where in FORTRAN, to call a procedure, special syntax was used, while in comparison to newer languages like C and Java only required the name of the procedure and any parameters to be passed in (Ben-Ari, 1996).

In concurrent programming, we can see the difference in concurrency across different programming languages. In Java, a thread can be called using a thread name and with the 'start' call, while Algol 68 needs threads to be constructed upon the 'cobegin' statement, before being executed, and then they need to be terminated before the 'coend' statement. In contrast, Java does not have such statements.

It is also to be noted that object-oriented programming (OOP) came about when in the era of FORTRAN and COBOL, there was a need to handle larger programs, showing the trend and needs of languages as programs grow in complexity and size. This paper also poses an interesting stance: Do we need more programming languages, with the abundance of languages already available in the modern day? And is there a need to reinvent the wheel? Chowdhary (2020) believes no one language is a one-size-fits-all, with the evolution of multi-cores, cloud computing and distributed systems. Thus, programming languages need to continue to develop and evolve, as the needs of programs are contextual in nature. Goyal et al (2020) also agrees that more user-friendly languages like python are less complex compared to older languages like C and thus shows the trend and need of simplicity in programming languages.

A study done by Swacha (2023) researched on the evolution of languages used in education in the past 5 decades, and they discover that Java remains popular since the 1990's, Python having a steady popularity rate, Pascal having a decrease in relevance since the 1980's, and assembly language surprisingly remaining a popular language in education.

CONCLUSION

In conclusion, this literature review covered key programming language concepts, which were syntax, semantics, data types, control structures, subprograms and (type of programming). This review also discussed the trends in programming language concepts across programming languages in different eras, and the advancements as to why evolution happens throughout the decades. In the discussion it also covered some controversies regarding programming concepts and challenges regarding them. Further research should be done on hybrid dynamic and static typing, as currently no language exists that can provide the advantages of both. Another possible area of interest in delving into is why and how a programming language concept, or rather feature, can be deemed ‘obsolete’, or outdated. All in all, advancements into programming languages and their concepts will most likely never cease, if only minor adjustments in newly created languages.

REFERENCES

Ben-Ari, M. (1996). *Understanding programming languages*. Wiley Chichester.

Cardelli, L., & Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, *17*(4), 471–523. https://doi.org/10.1145/6041.6042

Chowdhary, K. (2020). On the evolution of programming languages. *arXiv preprint arXiv:2007.02699*.

Goyal, J., Singla, A., Gupta, A., Malvika. (2020). Why there are So Many Programming Languages? *International Journal of Research in Engineering, Science and Management*, *3*(1), 515-517.

Hemmendinger, D. (2003). Concurrent programming. In *Encyclopedia of Computer Science* (pp. 439–445). John Wiley and Sons Ltd.

Jean-Pierre, J., & Mitsuhiro, O. (1997). Abstract data type systems. *Theoretical Computer Science*, *173*(2), 349-391. https://doi.org/https://doi.org/10.1016/S0304-3975(96)00161-2

Linda Weiser, F. (1992). From Babbage to Babel and beyond: A brief history of programming languages. *Computer Languages*, *17*(1), 1-17. https://doi.org/https://doi.org/10.1016/0096-0551(92)90019-J

Meijer, E., & Drayton, P. (2004, 01). *Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages*

PRISMA 2020 flow diagram — PRISMA statement. (n.d.). PRISMA Statement. https://www.prisma-statement.org/prisma-2020-flow-diagram

Strachey, C. (2000). Fundamental Concepts in Programming Languages. *Higher-Order and Symbolic Computation*, *13*(1), 11-49. https://doi.org/10.1023/A:1010000313106

Swacha, J. (2023). *Programming Languages in Education: 50 Years of Evolution as Evidenced by Literature* Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2, Toronto ON, Canada. https://doi.org/10.1145/3545947.3576330

Yasumura, M. (1977). Evolution of loop statements. *SIGPLAN Not.*, *12*(9), 124–129. https://doi.org/10.1145/954604.954615