COMPARATIVE ANALYSIS OF SOME PROGRAMMING LANGUAGES

Oguntunde, Bosede Oyenike
Department of Mathematical Sciences, Redeemer's University, Mowe Ogun State, Nigeria

Abstract
Programming languages are used for controlling the behavior of computer machines. Several programming languages exist and new are being created always. These programming languages become popular with use to different programmers because there is always a tradeoff between ease of learning and use, efficiency and power of expression. In this work we examine six programming languages two from different groups of scientific, non scientific and object oriented programming languages. We present an algorithm for performing combination and permutation to implement the comparison. Two parameters, the memory consumption and running time requirement are tested and objected oriented programming languages perform better in term of their running times although same could not be said of them in term of memory requirements.

Key Words: Combination, permutation, running time, memory requirement

1. Introduction

A programming language is an artificial language designed to express computations that can be performed by a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine, to express algorithms precisely, or as a mode of human communication.

Most programming languages describe computation in an imperative style, i.e., as a sequence of commands, although some languages, such as those that support functional programming or logic programming, use alternative forms of description.

Traits often considered important for what constitute a programming language include:

- *Function and target:* A *computer programming language* is a language used to write computer programs, which involve a computer performing some kind of computation or algorithm and possibly control external devices such as printers, disk drives, robots, and so on.

- *Abstractions:* Programming languages usually contain abstractions for defining and manipulating data structures or controlling the flow of execution. The practical necessity that a programming language support adequate abstractions is expressed by the abstraction principle; this principle is sometimes formulated as recommendation to the programmer to make proper use of such abstractions.
- *Expressive power:* The theory of computation classifies languages by the computations they are capable of expressing. All Turing complete languages can implement the same set of algorithms.

1.1 Syntax and semantic

All programming languages have some primitive building blocks for the description of data and the processes or transformations applied to them. These primitives are defined by syntactic and semantic rules which describe their structure and meaning respectively.

The syntax of a language describes the possible combinations of symbols that form a syntactically correct program. The meaning given to a combination of symbols is handled by semantics (either formal or hard-coded in a reference implementation). Since most languages are textual, this article discusses textual syntax.

The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms. For compiled languages, static semantics essentially include those semantic rules that can be checked at compile time. Examples include checking that every identifier is declared before it is used (in languages that require such declarations) or that the labels on the arms of a case statement are distinct.

1.2 Language processors

Generally, computers do not understand high-level languages. To make computer understand the program written in high-level language a translator will be required to translate the program into its machine language equivalent. Such translators are called Language processors. Language processors are of three classes namely: Assembler, Interpreter and compiler.

Assembler: it translate source program written in assembly code to machine code object programs.

Interpreter: it translates a high-level language one statement at a time and carries out the execution of the statement before proceeding to the next statement.

Compiler: it translates a program written in high-level language into its machine language equivalent.

The function of a language processor is to check for the syntactic and semantic correctness of statements in a program. This is done before the actual translation into machine language.

The subsequent chapters shall discuss in full the three programming languages: Scientific, non-scientific and object-oriented programming languages using one language as an illustration for each.

1.3 Our Objectives

Typically, we are interested in analyzing the "efficiency" of a given programming language. This involve determining the quantity of computer resources consumed by the language; measurable resources include the amount of memory (memory space) required by a language and the amount of computational time (running time) required by a programming language. We evaluate the performance of some programming languages using performance parameters such as running time and memory allocation.

2. Programming languages

Programming languages are grouped into three major categories: scientific, non-scientific and object oriented programming languages.

2.1.1 Scientific programming language

Scientific programming languages are procedural languages. They are command driven or statement oriented languages. Their basis concept is the machine state. For scientific programming language, FORTRAN and PASCAL would be used illustration.

2.2 Non-scientific programming language

Non-scientific programming languages are also procedural languages. They are closer and similar to most of human language. Basic and COBOL will be examined in this class.

2.3 Object oriented programming language

Object-Oriented Programming (OOP) is a method of designing and implementing software. OOP languages incorporate not just language syntax and compiler, but an entire development environment. These include a significant library of well-designed, easy to use objects.

It is a new way of thinking about problem solving with computers. An OOP language should contain the following seven qualities [2,3].

- The support of modules
- Module must support data abstraction
- Automatic memory management
- All data are defined by abstract data type based modules
- Ability to extend existing classes
- Support of polymorphism and dynamic binding
- Multiple inheritances.

We examine c++ and java programming languages in this class.

Several works have been done in the area of comparisons of programming languages. Jason Voegele presented a personal evaluation of some common programming languagesComparison between highly object based Java and non-object oriented by Benchmarking [1]. Comparison of programming languages by Alain [16]. Jason Voegele [14] presented a analytical comparison of nine different languages titled “Evaluation and comparison of programming languages”. The work was intended to provide very high-level information about the respective languages which are Eiffel, Smalltalk, Ruby, Java, C#, C++, Python, Perl and Visual Basic. Table 1 below shows the evaluation and comparison of some programming languages by Jason Voegele. This work however attempt to compare 6 different programming languages in three categories of scientific, non-scientific and object oriented programming languages.

Section one presents an introduction into the work, section two gives an overview of programming languages while permutation and combination algorithms are presented in section three. The implementation and data analysis are presented in section four and section five concludes the work.

Table 1. Evaluation and Comparison of Programming Languages

	Eiffel	Smalltalk	Ruby	Java	C#	C++	Python	Perl	Visual Basic
Object-Orientation	Pure	Pure	Pure	Hybrid	Hybrid	Hybrid / Multi-Paradigm	Hybrid		Partial Support
Static / Dynamic Typing	Static	Dynamic	Dynamic	Static	Static	Static	Dynamic	Dynamic	Static
Generic Classes	Yes	N/A	N/A	No	No	Yes	N/A	N/A	No
Inheritance	Multiple	Single	Single class, multiple "mixins"	Single class, multiple interfaces	Single class, multiple interface	Multiple	Multiple	Multiple	None
Feature Renaming	Yes	No	Yes	No	No	No	No	No	No
Method Overloading	No	No	No	Yes	Yes	Yes	No	No	No
Operator Overloading	Yes	Yes?	Yes	No	Yes	Yes	Yes	Yes	No
Higher Order Functions	Agents (with version 5)	Blocks	Blocks	No	No	No	Lambda Expressions	Yes (???)	No
Lexical Closures	Yes (inline agents)	Yes (blocks)	Yes (blocks)	No	No	No	Yes (since 2.1)	Yes	No
Garbage Collection	Mark and Sweep or Generational	Mark and Sweep or Generational	Mark and Sweep	Mark and Sweep or Generational	Mark and Sweep or Generational	None	Reference Counting	Reference Counting	Reference Counting
Uniform Access	Yes	N/A	Yes	No	No	No	No	No	Yes

	Eiffel	**Smalltalk**	**Ruby**	**Java**	**C#**	**C++**	**Python**	**Perl**	**Visual Basic**
Class Variables / Methods	No	Yes	Yes	Yes	Yes	Yes	No	No	No
Reflection	Yes (as of version 5)	Yes	Yes	Yes	Yes	No	Yes	Yes?	No
Access Control	Selective Export	Protected Data, Public Methods	public, protected, private	public, protected, "package", private	public, protected, private, internal, protected internal	public, protected, private, "friends"	Name Mangling	None	public, private
Design by Contract	Yes	No	Add-on	No	No	No	No	No	No
Multithreading	Implementation-Dependent	Implementation-Dependent	Yes	Yes	Yes	Libraries	Yes	No	No
Regular Expressions	No	No	Built-in	Standard Library	Standard Library	No	Standard Library	Built-in	No
Pointer Arithmetic	No	No	No	No	Yes	Yes	No	No	No

	Eiffel	**Smalltalk**	**Ruby**	**Java**	**C#**	**C++**	**Python**	**Perl**	**Visual Basic**
Language Integration	C, C++, Java	C	C, C++, Java	C, some C++	All .NET Languages	C, Assembler	C, C++, Java	C, C++	C (via DCOM)
Built-In Security	No	No?	Yes	Yes	Yes	No	No?	Yes (perlsec)	No
Capers Jones Language Level*	15	15	N/A	6	N/A	6	N/A	15	11

(Adapted from Jason Voegele, programming language comparison [14]) Based on number of source code lines per function point.

3 Permutation and combination algorithm

An algorithm can be defined as a step by step method of solving problems. The algorithm for the permutation and combination program is given bellow:

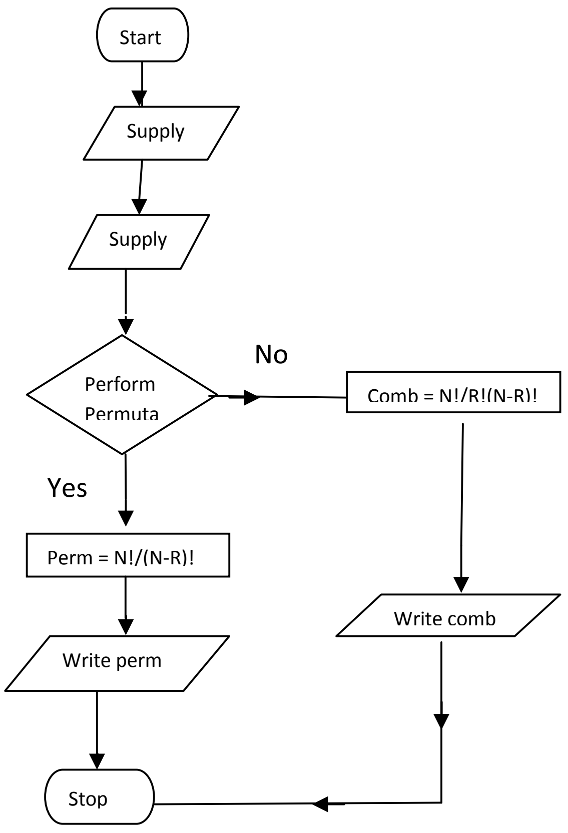

Fig.1 Flowchart

Algorithm

1. Supply the bigger positive integer as N

2. Supply the smaller positive integer as R

3. IF Permutation Then

Permutation = N! /(N-R)!

Else combination = N!/R!(N-R)!

5. Print permutation / combination

6. Exit.

4. System implementation / analysis of data

Different positive integer values were used in running and executing of the program. Each running time and memory requirement were recorded.

Illustration $^{20}P_8$ where N = 20 and R = 8

Using the permutation formula N! / (N-R)!

20! / (20 - 8)! = 20! /12!

= 5079110400

Combination $^{20}C_8$

Using the combination formula N!/R!(N-R)! with N= 20 and R= 8

= 20! /8! (20-8)! = 20! /8! * 12!

= 125970

Analysis

From our experiment, a set of values were obtained which give the running time and memory utilization of different programming languages for performing combination and permutation for different positive integers. We present these programming languages against their requirements in term of memory utilization and running times. The memory requirement for the source code measured in bytes is plotted against each of the selected programming languages and the running time requirements measured in seconds is also plotted against the programming languages.

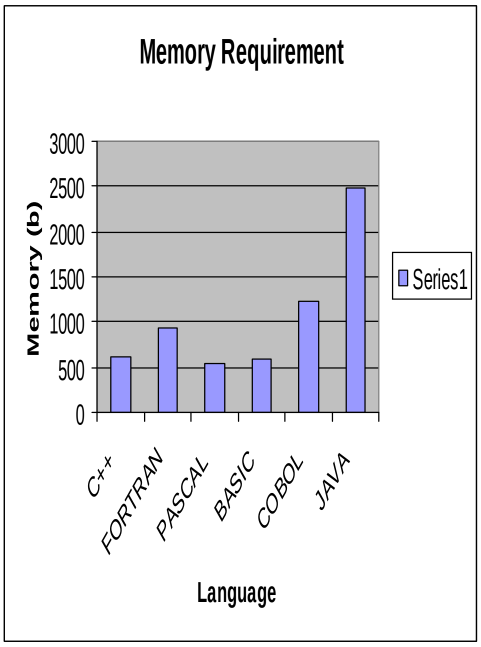

Fig.2 Memory requirements

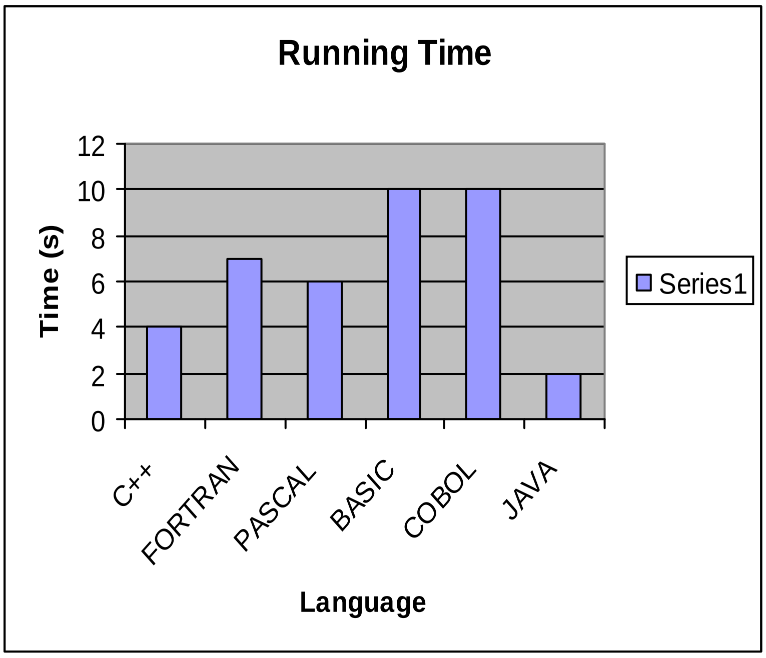

Fig.3 Running Time Requirements

4.1 Discussion

From the result above, it could be observed that Object-Oriented program has the smallest running time followed by scientific program then Non-scientific.

With the Object Oriented programming languages, JAVA program is faster than C++, although it required more memory allocation than all other programming languages.

With the Scientific programming language, PASCAL is faster than FORTRAN but they both have the same memory space requirements. Lastly, in Non-scientific programming language, BASIC has lesser running time compare to COBOL and it also required less memory space allocation compared to COBOL.

5. Conclusion

This work presents a comparison of some programming languages; we present a combination and permutation calculator written in all the languages to measure their performance. To conclude the write-up it is note-worthy that there are many other programming languages that are all improvement from the old ones. As different programming languages are being produced, two qualities (runtime and memory space) are being considered most. In reality, the fastest are not always the smallest, so tradeoffs between these two important qualities are being maintained.

Thus, Object Oriented Programming Language was designed to be a good tool for building a wide variety of systems and allow a wide variety of ideas to be expressed directly. Not everything can be expressed directly using the built-in features of a language. In fact, that isn’t even the idea. Language features exist to support a variety of programming style and techniques. Consequently, the task of learning a language should focus on mastering the native and natural styles for that language, not on the understanding of every little detail of all the language features.

Though our research is based on comparison between scientific, non-scientific and object oriented program using a program that compute the permutation and combination of n integer. Many researches can still be done on this topic using other program to illustrate. Likewise, there can be comparison of programming languages through the features of the languages.

References:

Benchmarking, Comparison between high object based java & non object oriented, (doi.wiley.com//0.1002/cpe.658)

Bertrand Mayer (1993). What is an Object-Oriented Environment? Five Principles and their Application, in Journal of Object-Oriented Programming, Volume 6, Number 4, July-August 1993, pages 75-81.

Bertrand Mayer (1994). An Object-Oriented Environment: Principles and Application, Prentice Hall, 1994.

John R. Hubbard, 1996, Schaum's outline of Theory and Problems of programming with C++, published by Mc Grew-Hill book Singapore.

Martine Robert Cecil, 2003, UMI for java programmers, Printice Hall.

Pius Ezeamii, 2005, Computer science for beginners, published by Infomedia, Lagos.

Rajaramon, H. V. Sahasrabuddhe, 1991, Computer programming in COBOL.

Seymour Lipschutz, 1996, Schaum's oytline of Theory and Problems of Programming with FORTRAN, Mc Grew-Hill, Singapore.

Yekinni A. Salimanu, Saheed O. Adelana, Wasiu A. Akanji, 2001, Introduction to comp. system & BASIC Programming, published by Oyaz-Gilgal company, Lagos.

http://www.threadvisors.com/lang cmparison.htm

http://userweb.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture25.html

http://www.mathsisfun.com/combinatorics/combinations-permutations.html

http://www.tutors4you.com/circularpermutations.htm

http://www.jvoegele.com/software/langcomp.html

http://en.wikipedia.org/wiki/Comparison_of_programming_languages

(http://b5cw.ercim.org/pub/b5ce.cgi/d283652/31-D'hondt-FP_2005_2006_SR_INRIA.pdf-.)

.