Programming Languages: A Comparative Study

Prashant Kulkarni, Kailash H D, Vaibhav Shankar, Shashi Nagarajan, Goutham D L

Abstract

In this paper we present a survey on programming languages- C++, Perl, Lisp and java. Our survey work involves a comparative study of these programming languages with respect to the following parameters like: Reusability, Portability, Reliability, Readability, Efficiency, Availability of compilers and tools, Familiarity and Expressiveness. At the end, a study of these languages has been made by considering- program length, programming effort, run time efficiency, memory consumption, and reliability as criteria, by running a single program by using the above said languages.

1. INTRODUCTION

The first high-level programming languages were designed during the 1950s. Ever since then, programming languages have been a fascinating and productive area of study. Programmers endlessly debate the relative merits of their favorite programming languages, sometimes with almost religious zeal. On a more academic level, computer scientists search for ways to design programming languages that combine expressive power with simplicity and efficiency.

The complexity of engineering software has increased dramatically in the past decade. In the early years most engineering applications were concerned solely with solving difficult numerical problems, and little attention was paid to man-machine interaction, data management, or integrated software systems. Now, computers are expected to solve a much wider variety of problems, particularly those in which numerical computations are less predominant. With the continuing increase in the variety, functionality, and complexity of engineering software, with its more widespread use, and with its increasing importance, more attention must be paid to programming language suitability so that rational decisions regarding language selection may be made.

A programming language should also be reasonably natural for solving problems, at least problems within its intended application area. For example, a programming language whose only data types are numbers and arrays might be natural for solving numerical problems, but would be less natural for solving problems in commerce or artificial intelligence. Conversely, a programming language whose only data types are strings and lists would be an unnatural choice for solving numerical problems.

C++

C++ is a fairly complicated object-oriented language derived from C. The syntax of C++ is a lot like C, with various extensions and extra keywords needed to support classes, inheritance and other OO features. C++ was originally developed as an extension to C, but quickly evolved into its language. Despite some of the flaws it has inherited from C, C++ is a very popular language for application development on Unix systems and PCs. The C++ programming language offers a very broad range of OOP features: multiple inheritance, strong typing, dynamic memory management, templates (generics), polymorphism, exception handling, and overloading. Some newer C++ systems also offer run-time type identification and separate namespaces. C++ also supports the usual features expected of an application language: a variety of data types including strings, arrays and structures, full I/O facilities, data pointers and type conversion. The C++ Standard Template Library (STL) offers a set of collection and abstract data type facilities. Because it is derived from C, C++ has a number of features that support unsafe and defective software. The more recent C++ standards do support safe casts, but this feature is not yet universally available or employed. Also, C++ has dynamic memory allocation, but does not have garbage collection; this allows programs to misuse and leak memory. C++ also supports dangerous raw memory pointers and pointer arithmetic. These low-level facilities are useful in some situations, but can increase the time needed for software development. Efforts at unifying the C++ language were begun in 1989. C++ was finally standardized by the ISO and ANSI in November, 1997. Information on C++ is widely available on the WWW, but language has no official home on the Web. Many C++ implementations exist, some of them follow the old tradition of translating C++ into C, while others are native compilers. A few free C++ compilers exist, the most notable of which is the GNU C/C++ compiler, GCC.

Java

Java is a simple, portable object-oriented language designed by

research staff at Sun Microsystems. The feel of the Java language is fairly similar to that of C++, but it also borrows ideas from Modula-3, Mesa, and Objective-C. The feature set of Java is fairly broad: it has inheritance, strong type checking, modularity (packages), exception handling, polymorphism, concurrency, dynamic loading of libraries, arrays, string handling, garbage collection, and a pretty extensive standard library. The newest version of the language, Java 1.6 (Java Platform 6), includes nested classes, persistence, and reflection as well as many additional standard libraries. The fundamental structural component of a Java program is the class. All data and methods in Java are associated with some class, there is no 'global' data or functions as in C++. Classes can be members of packages; package and class membership help determine scope and visibility of data and methods. Java does not include features that its designers felt would compromise the simplicity or safety of the language, so Java has no true pointers, no true multiple inheritance, no operator overloading, and no macro preprocessor. The lack of multiple inheritance could have been a serious shortcoming, but Java does support the definition and inheritance of multiple stateless "interfaces", which serve for most areas where multiple inheritance might be desired. Java also has no facility for generic functions, but since the language imposes a rooted class hierarchy (all object classes inherit from the root class 'Object'), any need for generics is greatly reduced. The Java standard library packages include extensive I/O facilities, a comprehensive GUI toolkit, collection classes, date/time support, cryptographic security classes, distributed computation support, and system interfaces. Java is typically compiled to platform-independent byte-codes. These byte-codes must be interpreted by a Java Virtual Machine (JVM), which may choose to compile the byte-codes further into native machine instructions. There is a strict definition of the Java byte-code file format, the .class file format, which ensures portability of compiled Java classes. In additional to normal application development, Java is used to develop embedded programs, called 'applets', for web browsers and other Java-enabled platforms. This capability is an important part of Java, and the standard library packages include a security manager to restrict the capabilities of Java applets. These applet facilities were important to Java's widespread adoption and popularity. Commercial Java compilers and development environments are readily available; among the most popular are products from Symantec and Microsoft. Java-soft supports and distributes a 'reference' Java implementation known as the JDK, it is free.

Lisp

Lisp is Functional or lambda-based languages and it is extremely rich and powerful programming language that has enjoyed continuous use and popularity since the mid-1960s. Typically, Lisp programming systems are interpreters, but compilers are also commonly used. The Lisp language is founded on the representational power of "S-expressions", and employment of functional composition and recursion. Lisp is a weakly typed language with excellent support for reflection and on-the-fly code generation and interpreting. The language's extreme flexibility, expressive power, and its ability to treat code as data, made it the undisputed king of Artificial Intelligence research for all of the 1970s and 1980s. All Lisp implementations since the late 1960s have offered a set of programming features tough to equal in any language, even today: macros, string handling, recursion, closures, reflection, packaging, arrays, and extensive IO facilities. Modern Lisp systems support object-oriented programming, database access, GUI construction, and all other forms of general-purpose programming. Lisp is the second-oldest high-level programming language in widespread use today; A very mature language, Lisp is extremely well-documented, and the most widespread dialect, Common Lisp, is also codified by ANSI and international standards.

Perl

Perl is an interpreted scripting language with extensive facilities for data manipulation and rapid application development. Perl is basically block-structured, but also supports object-oriented programming. "Perl" stands for "Practical Extraction and Reporting Language," a reference to the purpose for which the Perl interpreter was originally created: system administration and data reduction. Perl has gone through several major evolutionary phases. The current language version is Perl5, but some pockets of Perl4 use still exist. Perl5 is backward-compatible with 4. data types: strings, numbers, lists, associative arrays, references, globs data types: objects conventional math and arithmetic functions subroutines, variable argument lists dynamic memory handling with garbage collection extensive file I/O facilities extensive system interface support regular expression pattern matching and substitution very extensive data output formatting capabilities various loops and conditional constructs object definition and inheritance ,separable namespaces (packages), lexical and dynamic scope local variables, on-the-fly code evaluation and error handling

The Perl language does not support the traditional notion of records or structs. Instead, associative arrays (hashes) are provided to serve all such purposes. Similarly, Perl supports object-oriented programming, but does not stipulate an object storage format. In Perl5, code is parsed and compiled into very high-level bytecodes prior to interpreted execution. This approach, and extensive optimization of the Perl interpreter and run-time engine, allow Perl scripts to achieve very high performance. Perl currently does not support multi-threading, although efforts are underway to add this important feature. Perl is very popular in the UNIX community, and gaining acceptance in the Microsoft Windows developer community. There is only one Perl language system; written in C to be very portable, it runs on all UNIX platforms, 32-bit Windows, VMS, and many other systems. Perl is free. Books, tutorials, and on-line resources for Perl are widely available, and generally of good quality. Add-on modules and pre-built scripts for Perl are also widely available, with more being written all the time. Add-ons for Perl are so numerous and in such wide demand that an organized replicated archive system for them exists: the Comprehensive Perl Archive Network (CPAN). There is no international standard for Perl syntax. The language definition is informally set forth in Programming Perl, 2nd Edition, by Wall, Christiansen, and Schwartz.

2. CRITERIA FOR A GOOD LANGUAGE

To begin the language selection process, it is important to establish some criteria for what makes a language good. A good language choice should provide a path into the future in a number of important ways:

- Its definition should be independent of any particular hardware or operating system.
- Its definition should be standardized, and compiler implementations should comply with this standard.
- It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities.
- It should effectively support the application domain(s) of interest.
- It should support the required level of system reliability and safety.
- Its compiler implementations should be commensurate with the current state of technology.
- Appropriate software engineering-based supporting tools and environments should be available.

Effectively satisfying the above criteria is not easy, and it may require using different languages in different situations. However, as these points are violated, additional risk is involved with near-term development, as well as future technology changes. Key risks encountered as each of these criteria is violated are:

- If a language is not independent of a particular platform, portability is severely compromised. Hardware and software options are also limited, both for the original system and for future upgrades.
- If compiler implementations do not comply with a standard language definition, compiler-unique solutions are created. This also severely compromises portability, as well as options for future upgrades.
- To the extent that poor practices are used in software development, both development and debugging times will be extended, and poor code characteristics will make both testing and maintenance a nightmare.
- Poor support for the application domain will compromise the ease of development, as well as performance and readability characteristics of the code.
- If reliability is compromised, the system will not only perform below expectations, but it will also become much more costly across its lifetime. If safety is compromised, life and property will be endangered.
- An out-of-date compiler is inferior and difficult to use, producing substandard code which is difficult to create and maintain. It can also prohibit the use of key language features.
- The lack of appropriate automated development support compromises developer productivity and system quality.

3. CRITERIA OF LANGUAGE COMPARISON

We consider the following feature criteria:

Reusability: Does the language support effective reuse of program units? If so, the project can be accelerated by reusing tried-and-tested program units; it might also develop new program units suitable for future reuse. Relevant concepts here are packages, abstract types, classes, and particularly generic units.

Portability: Does the language help or hinder writing of portable code? In other words, can the code be moved from one platform to a dissimilar platform without major changes?

Reliability: Is the language designed in such a way that programming errors can be detected and eliminated as quickly as possible? Errors detected by compile-time checks are guaranteed absent in the running program, which is ideal. Errors detected by run-time checks are guaranteed to cause no harm other than throwing an exception (or at worst terminating the program),which is second-best. Errors not detected at all can cause unlimited harm (such as corrupting data) before the program crashes. While reliability is always important, it is absolutely essential in safety-critical systems.

Efficiency: Is the language capable of being implemented efficiently? Some aspects of object-oriented programming entail run-time overheads, such as class tags and dynamic dispatch. Run-time checks are costly (although some compilers are willing to suppress them, at the programmer's own risk). Garbage collection is also costly, slowing the program down at unpredictable times. Interpretive code is about ten times slower than native machine code. If critical parts of the program must be highly efficient, does the language allow them to be tuned by resort to low-level coding, or by calls to procedures written in a lower-level language?

Readability: Does the language help or hinder good programming practice? A language that enforces cryptic syntax, very short identifiers, default declarations, and an absence of type information makes it difficult to write readable code. The significant point is that code is read (by its author and other programmers) more often than it is written.

Availability of compilers and tools: Are good-quality compilers available for the language? A good-quality compiler enforces the language's syntax and type rules, generates correct and efficient object code, generates run-time checks (at least as an option) to trap any errors that cannot be detected at compile-time, and reports all errors clearly and accurately. Also, is a good-quality integrated development environment (IDE) available for the language? An IDE enhances productivity by combining a program editor, compiler, linker, debugger, and related tools into a single integrated system.

Familiarity: Are the available programmers already familiar with the language? If not, is high-quality training available, and will the investment in training justify itself in future projects?

Expressiveness: This factor reflects the ability of a language to express complex computations or complex data structures in appealing, intuitive ways.

4. EVALUATION OF PROGRAMMING LANGUAGES

C++

Its definition should be independent of any particular hardware or operating system. As with C++ is completely independent of any particular hardware or operating system.

Its definition should be standardized, and compiler implementations should comply with this standard. As with C++ has followed the standardization path of most languages. First, the language was created and used. As its popularity grew, it began to spawn a number of different dialects. Then, C++ started a standardization process, with the main core of the language being standardized. Because C++ is a relatively new language, its standardization process is not yet complete. It is expected that the C++ standard will be used much as the C standard, and it will be common for compiler implementations to support standard C++ with additional, system-dependent features. This will continue to result in the creation of much non-standard C++ code.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities. Unlike C, the structures and object-oriented features of C++ provide support for the concepts of encapsulation and data abstraction.

It should effectively support the application domain(s) of interest. As with C++ has proven to be a very versatile language, supporting any domain in which it has been tried.

It should support the required level of system reliability and safety. Reliability is supported by the object-oriented features of C++. Safety-critical systems, those on which human life may depend, are not effectively supported by C++ because of its lack of support for software engineering technology in its C subset.

Its compiler implementations should be commensurate with the current state of technology. Because of the immense popularity of C++, its compilers continue to be improved using current technology.

Appropriate software engineering-based supporting tools and environments should be available. Again, because of its popularity, a wide variety of supporting tools and environments is available for C++ development.

Readability: Although it is possible to write C++ code that is understandable, and the object-oriented nature of C++ is supported with understandable syntax, it is not common practice to use a verbose, understandable style for C++ any more than for C. C++ still provides the cryptic C shortcuts that run counter to clarity, and they are commonly used.

Maintainability: When C++ is being used to create object-oriented code, the programmer has good object-oriented features to facilitate maintainability. However, the C problem of little inherent support for maintainability still remains in other C++ language features.

Mixed language support: C++ will readily use object files produced by any language compiler as it composes an application. This is easy because C++ requires no consistency checking among these separate files. While that makes the object files easy to use, it does not provide specific support for properly interfacing the languages or for verifying correct exchange of data across the established interface. C++ improves on C with better language constructs for facilitating language interfacing.

Portability: C++ does not yet have an existing standard, but, when it does, it will probably not alter the C characteristics in this respect. Common practice will not necessarily adhere to the standard. However, C++ does encourage the encapsulation of dependencies, a feature which facilitates portability. C++ tools and tool sets are also widely available on many platforms.

Reliability: C++ improves considerably on the language characteristics of C for supporting reliability with features such as encapsulation, as well as improved expression.

Reusability: Support for reusability requires support for code clarity, encapsulation, maintainability, and portability. C++ provides much more inherent support for these characteristics.

Standardization: C++ is in the process of being standardized by both ANSI and ISO. However, once completed, there is no reasonable expectation that a C++ compiler will follow the standard without including additional features.

Support for modern engineering methods: C++ was created to support object-oriented programming, which provides support for encapsulation and data abstraction. This makes its software engineering support rather one-dimensional, but still substantial.

Java

Its definition should be independent of any particular hardware or operating system. Java is completely independent of any particular hardware or operating system.

Its definition should be standardized, and compiler implementations should comply with this standard. Java has had an unusual road to standardization. As with most languages, Java was developed and used before any attempt to get it officially standardized. However, many of the touted benefits of Java are lost unless a standard is strictly followed. Sun Microsystems, who developed Java, has dealt with this very effectively by making Java and its accompanying development tool kit freely available on the Web. Although Java has just recently begun the process of getting standardized through ISO [Sun 97], virtually all Java implementations follow the Sun de facto standard.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities. Java was developed largely because C++ could not effectively meet this criterion. Hence, although Java uses C++ syntax and it is object oriented, it is not similar to C++ in other substantial ways. Java both discourages and prohibits poor practices. Maintenance is supported by these characteristics.

It should effectively support the application domain(s) of interest. Java was developed specifically to support WWW applications. However, it is also a general-purpose language. Although it is still a very young language, it has proven to provide good support for any domain in which it has been tried.

It should support the required level of system reliability and safety. Java provides many features which support system reliability. For safety-critical systems, those on which human life may depend, no current language is entirely satisfactory. Java has not been around long enough to be studied for

suitability for safety-critical systems. However, since it does not provide formal analysis features, at best it would require a combination with mathematical specification, rigorous analysis, and formal proofs of correctness before it could be appropriate for use in safety-critical systems.

Its compiler implementations should be commensurate with the current state of technology. Java is very sophisticated for such a new language, and it is driving some of the current technology trends. Its Sun compiler implementation is commensurate with current technology.

Appropriate software engineering-based supporting tools and environments should be available. Current Java tool kits contain primarily tools to support code creation, although some software engineering-based tool kits are beginning to appear.

Readability: Java is strictly object oriented, so its form is very well defined. The code suffers somewhat from the cryptic C syntax forms.

Maintainability: Many features of Java support maintainability, such as those which support code clarity, encapsulation, and object orientation. Object-oriented capabilities can have both good and bad effects on maintainability, but, if used properly, object-oriented programming will improve maintainability.

Mixed language support: Java provides for interfacing with other languages by providing wrappers around the code from the other languages.

Portability: Java was built for complete portability. Its compiler produces source code in a platform-independent bytecode. The bytecode is then translated at runtime into native machine code for the given platform.

Reliability: Java requires the specification of information, the omission of which can make a program unreliable, such as type specifications.

Reusability: Java supports reusability with language features supporting code clarity (making code understandable), encapsulation (making code adaptable), maintainability, and portability.

Safety: Java was not developed for safety-critical systems, and its capabilities in that area are unproven.

Standardization: Java is in the process of ISO standardization. Nevertheless, it is currently a very effective de facto standard, and it is reasonable to expect implementations to follow the standard.

Support for modern engineering methods : Java was developed explicitly to support many software engineering principles, including its support for reliability, maintainability, and portability.

LISP

Its definition should be independent of any particular hardware or operating system. Lisp works on pretty much all operating systems.

Its definition should be standardized, and compiler implementations should comply with this standard. In this regard, Lisp is somewhat lacking when compared to other languages. Lisp has too many implementations most notably scheme and common lisp. Common lisp itself has a plethora of implementations. For a beginner, its often difficult to chose the implementation because of so many choices.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities. Common Lisp has the common Lisp Object System(CLOS) which supports OOP in lisp.

It should effectively support the application domain(s) of interest. Lisp was initially meant to be a language for AI research. It still has a lot of potential in this area. In other areas, it is slightly lacking due to the unavailability of good libraries.

It should support the required level of system reliability and safety. Lisp has very good mathematical foundations. It is used a lot for verifying safety critical systems.

Its compiler implementations should be commensurate with the current state of technology . Though lisp is an extremely old language, it is also easily extensible due to its powerful ‘code is data’ philosophy. For example, when the concept of OOP came up, it was added to the common lisp in no time.

Appropriate software engineering-based supporting tools and environments should be available. Though lisp provides good

tools in this regard, it is still not upto par with the other languages.

Readability: Lisp code is very readable. All statements begin and end with parantheses. This makes it easy for the compiler too. But it can be a slight problem for the beginner.

Maintainability: Leaving out common lisp's CLOS, lisp is mostly a functional language. Although functional languages are not as good as object-oriented ones at maintainability, lisp is an exception due to its flexible nature and clean code.

Mixed language support: Lisp provides for interfaces to other languages. Clojure is a lisp dialect targeting JVM.

Portability: Lisp is portable across many operating systems

Reliability: Lisp is very reliable. Lisp has good type inference which eliminates errors at compile time.

Reusability: Lisp programming is bottom-up. Working bottom-up is also the best way to get reusable software. The essence of writing reusable software is to separate the general from the specific, and bottom-up programming inherently creates such a separation.

Standardization: There are way too many common lisp implementations. However, these implementations have standards of their own (notably ANSI common lisp) which they adhere to.

Support for modern engineering methods: Many modern lisp implementations support modern engineering method. Since lisp is an easily extensible language, it is very easy to add any new feature into lisp.

Perl

Its definition should be independent of any particular hardware or operating system. Perl is completely independent of any particular hardware or operating system.

Its definition should be standardized, and compiler implementations should comply with this standard. Perl began as a single man's project when Larry Wall wanted to develop a language which provided all the functionalities he needed. It has since evolved and now has fixed standards now. Perl has a GNU standard version which is used now. Other distributions of Perl exist but are not widely used.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities. Perl was developed to make development simple and quick. It prohibits poor practice. However, it is not widely used for software engineering technology.

It should effectively support the application domain(s) of interest. Perl was developed for rapid string processing, parsing and web development. It has lived up to its expectations and even delved into other application domains since its creation.

It should support the required level of system reliability and safety. Perl provides many features which support system reliability. For safety-critical systems, those on which human life may depend, no current language is entirely satisfactory. Perl has been studied for suitability for safety-critical systems. The Perl security man page indicates the possible bugs in Perl safety and precautions to be taken while developing sensitive systems.

Its compiler implementations should be commensurate with the current state of technology. Perl was developed for fast text processing and for server side scripting. The GNU version of its interpreter is widely accepted and freely available.

Appropriate software engineering-based supporting tools and environments should be available. Software Engineering based support tools are being studied in Perl. In fact, a recent book by Ditcher and Pease has been highly acclaimed for the possibilities of Perl's use in Software Engineering.

Readability: Perl is a structured language, but the resemblance of its commands to the English language makes it easily readable.

Maintainability: Perl supports the Object Oriented model as well as the structured programming approach. It bears resemblance to C++ but is not as strict as C++ in its syntax.

Mixed language support: Perl does not provide wrappers as Java does for other languages. However, it often acts as a glue and is used to call upon various different programs from the system command line.

Portability: Perl is highly portable. As it uses an interpreter, it is platform independent and can be used on any OS. It has been tested on over 70 Operating Systems successfully.

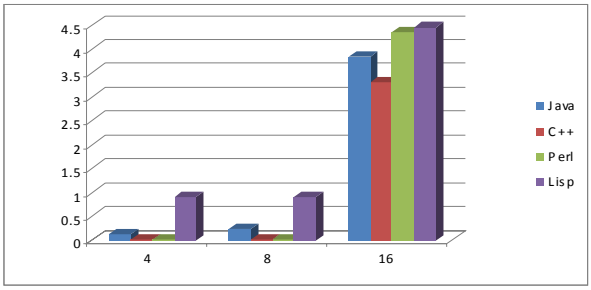
Reliability: Perl is flexible in its declaration of variables and can be unreliable in certain cases.

5. RESULTS

The same program (i.e. an implementation of the same set of requirements) is considered for each language. Hence, the comparison is narrow but homogeneous. Several different aspects are investigated, such as program length, programming effort, run time efficiency, memory consumption, and reliability.

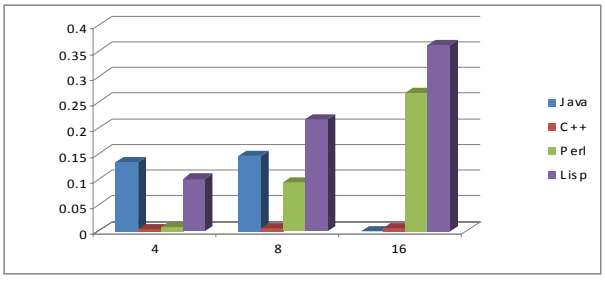
Evaluation of Tower of Hanoi programs for 4,8 and 16 Disk (execution time in seconds)

	4 Disk	8 Disk	16Disk
java	0.131	0.231	3.836
c++	0.005	0.006	3.307
Perl	0.007	0.011	4.349
lisp	0.899	0.903	4.454



Evaluation of Ackermann algorithm for input (3,2) ,(2,100) and (1,1000) m, n values (execution time in seconds)

	(3,2)	(2,100)	(1,1000)
java	0.134	0.145	Stack over flow
c++	0.004	0.005	0.006
Perl	0.008	0.095	0.268
lisp	0.101	0.217	0.361



RSA implementation

RSA is a public key cryptography and we implemented the same in four different languages. Our general feedback on that

- In Java the code is easy to write because of the packages available. Also our implementation was only encryption and decrpytion. But to generate primes in Java there are packages available.
- In C++ the code was not very difficult. It was difficult to configure the arithmetic precision math library and hence the prime size was restricted to that of int (32 bit) and n was 64 bit.
- In case of Perl the precision is lost like in c++. Hence max value till which we could implement was less (10 pow 10)
- In case of lisp the code was easy and it could handle large numbers without any packages unlike java where we used BigInteger class.

	CPU	Memory	Time	comment
Java	12.6	0.6	0.2	1024 bit using BigInteger and math library
C++	97.2	Negligible	34.5	Brute force code 32 bit
Perl	98.3	Negligible	2min 43s	Brute force code 32 bit
Lisp	56	0.8	2.143s	Easy to implement the efficient algorithm

Brainfuck Interpreter

Brainfuck is an esoteric programming language. It mainly contains eight commands. It is a turing complete language. It can be visualized as a array of elements on an semi infinite tape. The eight commands are 1) Increment the cell value(+) 2) Decrement the cell value(-) 3) Move to the left cell(<) 4) Move to the right cell (>) 5) Display the content of the cell (.) 6) Take

input (,) (7) & (8) to implement loops ([,]). We built a interpreter of this language as a task. These were the following observations

- In case of c++ because of the pointer arithmetic which is available we could implement the same very easily.
- In case of Java the implementation was not hard as the standard package like vector and stack
- In case of Scheme there was a problem due to lack of packages to handle vectors and pointers.
- In case of perl as well there was no problem. It used stack data structure to implement the loops.

	CPU	Memory	Time	Comments
Java	40.6	0.5	7.074	Large test case
C++	23.3	0.1	1.398	Large test case
Perl	97.9	0.3	3min 23s	Large test case. Very slow in the looping
Lisp	10.3	0.3	2.132	Own test case.. partial implemetation

Othello

Othello is a basic AI game. We have implemented a human vs computer game. The aim of the game is to max the no. of your colour pieces on the board. The rules are you need to place the coin in such a way that atleast one of the opponent coins is overturned. All the opponents coins which are in between your coins horizontally, vertically or diagonally will be overturned. The programming required is similar to that of a chess game. We try to construct a minimax search tree. Your analysis

- In case of Java we had more flexibility in the code and it was easy to implement the game becuase of the object oriented approach. We also implemented a GUI with the help of swing which is part of the standard jdk.
- In case of C++ the implementation was the same as that of java. But the GUI was slightly more harder as we needed to supplement it with QT. Hence it was not possible in a weeks' time.
- In Perl, the engine was fairly easy to design. However, passing of vectors in recursion was found quite complicated. The GUI was harder than that of Java and was not implemented in the specified week.

	CPU	Memory	Comments
Java	2.3	1.3	Memory consumed mostly due to applet
C++	2.0	Negligible	Command line interface
Perl	1.8	Negligible	Command line interface

Web Server

We build a very basic web server which can handle HTTP Get request. The server was single threaded.

- In java we provided a GUI but the request was taking time to be processed. It could handle images as well. Not just plain html.
- In C++ no GUI was provided but the request was easier to handle. The images were not getting loaded properly.
- In Perl it was lot more easier to handle the requests. It could handle image request as well.
- In Lisp the code was a lot simpler. But the functionality was restricted as well.

	CPU	Comments
Java	2.7	Standard packages used. Easy to program
C++	2.0	Extra packages used. Not easy to set up socket streams
Perl	0.7	Standard packages used. Easy to program
Lisp	3.0	Standard packages. Very short and easy to program

6. CONCLUSION

JAVA is relatively new but it has made substantial leaps in recent history. JAVA was designed bottom up to be a safe object oriented language with a very strong focus on "what to do" rather than "how to do." Consequently, JAVA is highly standardized, strongly typed, and has a rich set of APIs that make it easy to write programs. However, JAVA is quite verbose. Further, one needs to really understand object oriented programming to make the most effective use of JAVA. For people used to programming in C/C++, JAVA might seem restrictive since it does not use pointers at all, and it does not provide the "raw" power of C/C++

C++ derives both its popularity and problems from C, since it is so deeply rooted in it. C++ adds several powerful features to C, such as templates and object orientation, but these are seldom of relevance to the novice programmer. However, C++ does get rid of many of C's problems. Consequently, very often C++ is used in place of C to teach procedural programming. The typical memory consumption of a script program is about twice that of a C++ program. For Java it is another factor of two higher.

Perl has been in existence for nearly two decades and has become immensely popular. It was created in order to reduce the shortcomings on any one language. It was developed using a mix of C/C++, Java, sed, grep, awk, Shell etc with an attempt to keep the best features of them all in one language. It is most useful in developing parsers and other fast text processing applications. It also has significant use in web development.

Lisp is Functional or lambda-based languages and it is extremely rich and powerful programming language that has enjoyed continuous use and popularity since the mid-1960s. Lisp is a weakly typed language with excellent support for reflection and on-the-fly code generation and interpreting. The language's extreme flexibility, expressive power, and its ability to treat code as data, made it the undisputed king of Artificial Intelligence research for all of the 1970s and 1980s. All Lisp implementations since the late 1960s have offered a set of programming features tough to equal in any language, even today: macros, string handling, recursion, closures, reflection, packaging, arrays, and extensive IO facilities.

It is important to make technology decisions at the right time and for the right reasons. Good business decisions provide good people with appropriate supporting tools so they can produce good products. When it comes to software development, dealing with tough language issues head-on is one requirement for today's visionary manager. When combined with other software engineering considerations, a good language decision can support the development of cost-effective software systems that, in turn, provide valuable, reliable business support.

7. REFERENCES

[1] "Programming language Design and Concept", David A. Watt, University of Glasgow

[2] "An Empirical Comparison of Seven Programming Languages", Lutz Prechelt, IEEE- 2000

[3] "An Empirical Study of Programming Language Trends", Yaofei Chen, Rose Dios, Ali Mili, and Lan Wu, *New Jersey Institute of Technology,* Kefei Wang, *State University of New York, Albany ,IEEE-2005*

[4] "Selecting a Programming Language for Your Project", David Naiditch *Raytheon Systems Company, IEEE- 1999*

[5] "Seven Deadly Sins of Introducctory Prograimmting Language Design", Linda Mclver , Damian Conlway, *Monash University, Victoria, Awstralia, IEEE- 1996*