C and Java: Structural Comparison

Java and C are two important programming languages that are the cornerstones of computer programming. Both have gained a large user base and developer community over the years. In this essay, we will try to understand the strengths and weaknesses of both languages by comparing Java and C languages in depth from several aspects. In the following chapters, we will discuss several properties of both languages such as lexical syntax, concrete syntax, abstract syntax, keywords, identifiers, constants, strings, operators, delimiters, data types and type safety, paradigms, memory management, readability, writability, reliability, orthogonality, cost, availability, efficiency and learning curve. We will compare these important factors. These comparisons will help us better understand the use of both languages and help us determine how to best choose our projects.

Lexical syntax defines the basic elements of a programming language. Java and C languages have many similar features in terms of lexical syntax. Both languages are case sensitive, meaning uppercase and lowercase are considered different characters. Also, both languages ignore whitespace and comments, have rules defining certain keywords and identifiers. The use of low-level features such as pointers in C and its unique structures such as structs make it different from many other languages and reduce people's familiarity with the syntax. Since the Java language implements the object-oriented programming paradigm, its lexical syntax is formed within the framework of these object-oriented programming principles and is more universal. The public keyword found in the Java language is also found in many object-oriented programming languages, and a developer who knows object-oriented programming can easily understand a code written in Java. In the other hand a programmer who do not know C language when sees structures such as structs, pointers in C code probably must do additional research to understand the C code. For these reasons and the fact that Java is designed as a higher-level language, Java offers a more regular lexical syntax than C.

Concrete syntax defines how a program should be written. Java and C differ greatly in terms of concrete syntax. C is a language with a lot of prominent symbols such as curly brackets, semicolons, and asterisks. Moreover, memory management must be done by the programmer, which causes additional code writing and code pollution. In Java, this is automatically provided by the garbage collector. Java offers a cleaner and more organized concrete syntax than the C language. Since it is an object-oriented language, its spelling rules are clear and readable. For example, to create a class that you simply created in Java in a similar way in C, you need to create it in two parts, using a struct, with a header file and a C file. At the same time, since Java is object-oriented and most languages today adopt object-oriented principles, Java's concrete syntax is more universal and understandable.

Abstract syntax defines the logical structure of a programming language. Both Java and C offer similar structures in terms of abstract syntax. The main reason for this is that Java was inspired by C++ and C++ was inspired by the C language, so we can say that the C language is the ancestor of Java. Both languages include basic structures such as variables, functions, loops, and conditional statements. However, while the C language has more low-level control and memory management features, Java handles these operations itself, providing a higher-level abstract syntax and providing the programmer with greater abstraction and automation.

Java and C have similarities in terms of keywords of both languages. Both languages have keywords that describe the basic program structure. For example, control and loop structures such as "if", "else", "while", "for", "do while" are similar in both languages. However, both languages also have their own unique keywords. While Java has keywords that support object-oriented programming such as "public", "class", "static", "enum", "record". In the other hand C has keywords that reflect the specifics of the language such as "struct", "union" and "typedef".

Identifiers in both languages follow similar rules. Identifiers represent the names of variables, functions, or classes. Both languages are case sensitive and identifiers must begin with a letter or an underscore. However, Java offers more standards and recommendations in that identifiers are more descriptive and encourage certain writing styles, such as variables and functions being "camelCase", classes being "UpperCase", and constants being "CAPITAL".

Java and C are similar in defining and using constants. Both languages support numeric and textual constants. However, Java offers more features in constants. For example, in Java, constants defined with the "final" keyword are marked as immutable and their references cannot be changed. This is not the case in C.

Java and C take different approaches to expressing strings. In Java, there are much more options for character arrays, for example, they can be processed using the String class, StringBuilder class or character array, whereas in C, strings are treated as character arrays and there are no functions other than a few built-in functions. In Java, String and StringBuilder classes have many functions such as concatenation, addition, search, substring creation on character arrays, and the programmer is more flexible on character arrays, which indicates that Java is a more functional language in this regard.

Both languages contain basic operators (arithmetic, comparison, logical, assignment, etc.). However, while C distinguishes between signed and unsigned arithmetic operators, Java does not make this distinction. This leads C to offer greater flexibility for low-level operations.

Java and C use separators for organizing code. Both languages use separators in similar ways, such as semicolons, parentheses, and curly braces.

Data types and type safety ensure that the code runs more smoothly, prevents errors, makes the code understandable, and reduces security vulnerabilities by using the correct data types in programs. It also improves performance and improves the overall quality of the software. Java offers strict type checking, and this is done to minimize errors caused by type incompatibility. In Java, the data type of each variable must be explicitly declared, and automatic type casting of variables between different types is strictly controlled. Conversion operations, such as converting a decimal number to an integer, are carefully managed to prevent possible data loss or errors. Java is therefore stricter in terms of type safety and offers more assurance to the programmer. The C language, on the other hand, takes a more flexible approach to type safety. C allows you to use the data types of variables more freely and leaves the conversion to the programmer. This can sometimes lead to errors and data loss because C expects the programmer to handle the conversion operations correctly. For example, when you convert a decimal number to an integer, this conversion process may cause data loss, but the C programmer needs to check this issue. So, in terms of type safety, Java offers a more strict and safe approach, while C takes a more flexible approach but requires careful use. Which language to choose may vary depending on the needs of the project, security requirements and programmer preferences.

Programming paradigms are the fundamental principles that guide and shape software development processes. Java is known as a language that supports object-oriented programming principles and has the advanced concepts of OOP. In Java, everything is organized around objects and classes, increasing the organization and reusability of the code. Inheritance between classes, multiple interface implementations, and other OOP concepts can be used in Java. The C language was not originally designed to provide OOP support. In C, low-level data structures and functions are prominent. However, structs and function pointers can be used to perform similar operations in C. OOP cannot be fully implemented in C because C does not have specific language features for OOP structures such as classes, objects, inheritance, and interfaces. The procedural nature of the C language is advantageous for system programming and high-performance applications by providing flexibility in providing low-level control, memory management, and hardware access. Because Java is non-procedural, lacks low-level control, and automates memory management, it can cause low-level optimizations in some applications or throttles in areas that need performance. For these reasons, Java fully supports the principles of OOP, while C is more limited in its OOP support and is used mostly for low-level programming. While Java is a more suitable option for large-scale object-oriented projects, C is preferred for low-level programs.

Memory management is a fundamental concept in programming languages and has a huge impact on the performance, security, and efficiency of software. Java offers automatic memory management with garbage collector. This makes it unnecessary for the programmer to deal with memory allocation and release. When objects are created in Java, if there are references pointing to these objects and an object is no longer used, Java's garbage collector automatically collects these objects. This protects against memory leaks and incorrect memory usage. C language does not offer automatic memory management. Memory allocation and release operations are the responsibility of the programmer. The C programmer must call functions such as malloc to allocate memory and release the memory when use ends. This requires careful memory management, and incorrect memory management can lead to memory leaks or other problems. Therefore, Java is safer in terms of memory management because the programmer does not have to take care of memory management. In C, memory management requires more programmer intervention and therefore must be done carefully. In terms of flexibility, the C language is far superior; you can hardly manage memory in Java, but in C, all memory is under the command of the programmer.

Readable code not only makes it easier for software developers to quickly understand and correct code, but also helps collaborative developers communicate more effectively. Java offers a clean and tidy syntax and uses blocks of code clearly separated by curly brackets. This makes the code more readable and understandable. Additionally, Java's object-oriented programming structure allows for better organization and understanding of the code. OOP tools such as class diagrams can also help better document and understand code. C language does not support OOP and is designed as a procedural language. For this reason, C codes are usually written top-down, as codes that run according to the order of certain steps. This can sometimes require reading through the entire code to understand the code, which can reduce readability. Especially in large projects, Java's regular and readable syntax and the organizational advantages of OOP contribute to making the code more understandable. Since C is a low-level language, it can contain more details, which can make the code more difficult to read.

Writability allows code to be written and developed quickly and efficiently, saving time and resources. As a low-level language, C provides more control and flexibility. However, this control and flexibility can complicate the writing process. Especially when working with arrays, operations such as calculating the length of arrays, memory allocation, pointer usage, and loops may be required. This can make the writing process more complicated. Java, on the other hand, increases writability as a higher-level language. Java's data structures and class libraries allow you to easily perform complex operations. For example, when working with arrays, operations such as merging, changing and deleting arrays are much easier by using classes and interfaces such as Array and List in Java. However, these conveniences of Java can sometimes lead to writing code without knowing what should be used where and negatively affect the efficiency of the software.

Reliability ensures that software works as expected and minimizes the risk of errors, which ensures that applications are reliable and stable. Java's strict type checking and automatic memory management minimize errors and increase reliability. In C, memory management and type checking may require more responsibility, so it may be more prone to errors. For example, if you have a char array of length 10, when you try to access it as array[10] in C language, you may get a segmentation fault error or you may get a meaningless or meaningful result instead of an error because in C language, memory management is completely in the hands of the programmer and you display whatever is in that memory, whereas in Java, array[10] When you try to access it as you will get an error like "Index 10 out of bounds for length 10". Likewise, data conversion in Java will give an error if it is not done in a controlled manner. For example, “int x = 2.5;” The line gives an error: “int x = (int)2.5;” It is expected to be done in a controlled manner. This situation does not exist in C, C deletes the float part in this example and assigns the value 2 into the variable. If the security situation in C is not handled carefully, it can lead to huge problems. For this reason, we can say that Java is a safer language.

Orthogonality refers to the ability of each feature in a language to be used independently and consistently. The orthogonality of the C language can increase the complexity of the language in some cases. Java tends to have a more consistent language structure, which makes the code more predictable. In Java, the orthogonality property can be handled through a user interface and database example. In Java, you can change the database and it does not affect the interface. Likewise, you can make changes to the interface, and it will not affect the database. In the C language, the orthogonality feature of the language may encounter some exceptions. For example, in C, structs that are not arrays can be returned from a function. A member of a struct can be void or any data type except a structure of the same type. An array element can be any data type except void. Such exceptions affect the orthogonality of the C language and can make the language difficult to learn and use.

In terms of cost, Java is higher than C. Java applications require more memory and processing power and are therefore more costly. C works more efficiently as a low-level language. The biggest reason why Java is costly is that everything is done with classes. First, the classes are loaded into memory, the objects in them are initialized, static and instance initializers, if any, are run and the program gets started. Additionally, the written codes are translated into bytecodes that the JVM can understand, and these are run on the JVM. For this reason, Java is much more costly. The codes written in C language are directly translated into machine language and run by the machine, which minimizes the cost.

Both languages are widely used, so there are abundant resources available in terms of documentation, libraries, and support, greatly increasing the availability of both languages. Codes compiled in Java are converted to bytecode and run on the JVM, which forms the basis of Java's "write once run anywhere" philosophy. Java codes can be run anywhere, regardless of platform. The C language is not such a platform-independent language, even its standard library may vary from platform to platform.

In terms of efficiency, C generally offers higher efficiency as a low-level language. In the C language, the memory is completely open to the use of the programmer and programmer can move on the memory as own wishes and make the changes own wants. In addition, since the code is converted directly into machine code, it can be run directly on the operating system or hardware, which greatly reduces the cost of memory usage. Java, on the other hand, is a higher-level language and uses more processing power and memory, the main reason for this is that everything is a class and runs on the JVM.

A learning curve in a programming language helps new developers quickly understand the language's capabilities and enables projects to be developed more efficiently. Because Java implements object-oriented programming principles, learning OOP concepts can be difficult for beginners. Concepts such as classes, objects, inheritance, interfaces may seem daunting at first. However, this does not mean that Java's learning curve is very steep, because many of the ideas used offer ways for beginners to abstract OOP concepts. For example, when you create a project, the project's class and main function are created automatically, and they also provide tasks such as automatic import of ideas. Thanks to these, daunting OOP concepts for beginners can be removed from the eyes of beginners. Additionally, Java has a large standard library. This provides developers with ready-made solutions. It allows some basic tasks to be performed easily. Java also has a very large community and a very extensive documentation, for example, books such as "head first design patterns" and "clean code", which are cited in the software world, use Java in their explanations. C may seem simpler at first because it is a low-level language. Basic programming structures, loops, conditional statements, and functions can be learned quickly. However, memory management, pointers and struct issues in C language can be very challenging for beginners. The learning curve may vary depending on factors such as a person's current experience, goals, and projects. For beginners, Java offers a smoother learning curve, but it can feel more challenging at first, whereas the C language seems easy up to a certain point, but after a point, the learning curve increases very steeply.

As a result, the fact that Java and C languages offer different design philosophies, usage scenarios and advantages show that both languages offer ideal options for different projects and requirements. Java can be preferred for large software projects, web applications and network programming due to its features such as platform independence, security, and automatic memory management and because it is a high-level language. In addition, Java's excellent implementation of OOP principles, a large developer community and a comprehensive standards library play a major role in Java's widespread use. The C language is an ideal choice for low-level hardware interaction, system programming, and applications where performance is critically important. C provides direct access to hardware and offers greater programmer control. It is frequently used in writing operating systems, drivers, and embedded systems.