CSE 262: Quiz #3  
Due October 21st, 2022 at 11:59 PM

The quiz has TWO questions. Please submit your answer by updating this file in the quizzes folder of your Bitbucket account, and then committing and pushing. You should use as much space as you want for each answer. Please be detailed in your answers. Remember: this quiz is worth 9% of your grade, and you will not receive very many points if you do not give detailed answers.

**Question 1:** In class, we discussed that a compiler for a type-safe language *should* know the shape of its data structures. Using this claim as a starting point, study the JSON file format. Then investigate the Google Protocol Buffer format. How do they address the same problem? How do they address different problems? What are the strengths and weaknesses of each?

Compilers for a type-safe language should know the shape of its data structures to maintain data truthfulness until they die. Therefore, when we transmit data to other applications, we need to preserve the data such as their value and types. Similarly, when we receive transmitted data from other applications, we must know what that data represents to interpret and store it correctly. Since there are many ways to store and interpret information, data formats are used to give the data that we send, such as bytes, its meaning so that it can be interpreted correctly.

Two data formats that are commonly used are JSON and Google Protocol Buffers. To start, I would like to explain the JSON and the Google Protocol Buffers formats. JSON is in a text data format that is human-readable, plain-text, uses key-value pairs, and uses a map data structure that is easy to understand. JSON has a base structure that requires no schema and is simply a message. JSON creates a pair with a key, which is the type, and the value, which is the content. If the key is a JSON object that contains many JSON types, it uses a map data structure to represent it. JSON type’s support only Strings, numbers, JSON objects, arrays, Booleans, and Null data types. On the other hand, Google Protocol Buffers uses a binary format created by Google to serialize and deserialize data between different services. The Google Protocol Buffer uses a schema that define the set of rules for the data structure and can support a wider range of data types.

These two data formats address some similar common problems. One common problem that they both address is the transfer of data between systems. These two formats can send their data through their serialization based on their data formats that can be interpreted and used by an accepting recipient. Another common problem that they both address is speed. Both of these two data formats are very efficient in encoding. JSON and Google Protocol buffer make use of effective encoding that is much faster and lighter in file size than XML.

Although these two data formats address some similar problems, they also individually address different problems. For example, JSON is primarily used in web development because its text data provides a way for data to be transferred while maintaining a form that is human-readable; this trait is especially beneficial in web development where humans need to be able to read to interpret the information. On the other hand, Google Protocol Buffer is in binary format, where the data is encoded in a bunch of 0s and 1 which is not human-friendly to read, but it is extremely efficient since the data is compressed better. The binary format allows data serialization and deserialization to be simpler, smaller, faster, and more maintainable than other data formats such as JSON and XML.

The strengths and weaknesses of JSON and the Google Protocol Buffer come from their data format designs and languages and systems that they support. In general, JSON with its text data format would be better for applications that need humans interacting with the data that is being sent and received. JSON’s human-readability benefit also makes debugging easier. On the other hand, the Google Protocol Buffer that has a binary format, which is much harder to read, is designed to be more efficient for processes that do not require a lot of human interaction with the data. Since the Google Protocol Buffer is much more efficient but not readable, it is used in many large internal services that interchange a lot of data where JSON would be much less inefficient for file size and speed. It is also important to note that JSON supports built-in functions and libraries in almost every language and system, but the Google Protocol Buffer supports only a limited number of languages. Furthermore, as mentioned before, JSON supports a limited number of type’s, which are Strings, numbers, JSON objects, arrays, Booleans, and Null. On the other hand, The Google Protocol Buffers can support a wider range of data types. In general, these two data formats have their own strengths and weaknesses and should be used depending on the goals and needs of the applications.

**--------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Question 2:** Many modern languages have built-in vector and unordered map (hash table) data types, which are capable of holding arbitrary data types. For example, even before the Go language had proper generics, it still supported generic vectors and unordered maps. Other common data types (double-ended queues, lists, ordered maps, queues, stacks, priority queues, etc...) are not built in. There are many possible reasons. Using your knowledge of data structures and what you have learned in programming languages so far, give at least two reasons why these other data types are less likely to have first-class support. Use different data structures to support your first and second reasons.

There are many possible reasons why common data structures, such as the ones listed, are not built into some programming languages. One of the primary reasons is that the common data structures would have to be very generic in order for them to handle all data types. Because of this, the standard libraries for the data structures that we import would have to become much bigger and incorporated directly in the programming language. This extension would make the programming language itself a lot bigger and harder to deploy into small environments and therefore it leads to less flexibility and simplicity. For example, the linked list library is not part of C but is part of Java. The C programming language is known to be faster than Java as well as smaller in size. This is because C does not have to provide first-class support for data structures like the linked list library, rather people build the LinkedList data structure themselves in a way that is efficient for them. In most cases in C, with has no overhead abstractions, people can make good enough data structures themselves to the point where there is no need for it to be in the standard library. This makes C much more flexible and easier to deploy in small environments. On the other hand, the Java programming language does handle first class support for the linked-list data structure. Because of this, they must incorporate syntax support, run time support, and much more, which could take a heavy toll on performance as well as a large increase in size for the programming language. This makes Java less flexible than C and harder to deploy in small environments. In summary, having built in data structures could took a toll on performance and feasibility on programs in a particular language for a small convenience.

Another major reason that common data structures are not built into some programming languages is because of its impacts on compiler efficiency. Some programming languages are supported by not a few, but many compilers that all interpret the same programming language somewhat differently. Therefore, having a built-in data structure in the programming language would limit the number of compilers that can support the programming language since some may not be able to interpret the data structure that would be built-in correctly. This would mean that it would be harder to write different compilers for the programming language. Having multiple different compilers helps with making the program more efficient depending on its tasks, fixing their own separate issues. Therefore, making it harder to create more compilers leads to less efficiency. For example, C has many compilers that may have different purposes. It could use the Intel compiler because it frequently compiles faster code, it can compile under Clang to get better error messages, and much more. On the other hand, Rust has only one compiler, which limits the number of options a user can choose from.

In addition to its impact on compiler efficiency, common data structures are not built into some programming languages because of the potential overheads and vulnerability that may be associated with having one design for a data structure. Depending on the data structure and its design, there could be overhead depending on the application because of the programming language’s interpretation. These overheads and vulnerability can be costly since there may be many people using the built-in data structure without knowing that there is an overhead/vulnerability toll on their application. On the other hand, in non-built int programming languages, there could be alternative fixes by reconstructing data structures differently for different uses. For example, in the case of a Hash Table, the C++ programming language allows one to choose between a pointer and the value to fix overhead problems in the Hash Table for linear time complexity. On the other hand, GO has a built-in Hash Table, because they know they are going to be using a lot of HashTable but has a vulnerability issue. Although GO also fixes the overhead problems for linear time complexity by condensing the keys and the way elements are reinserted into a bigger hash table, it creates a vulnerability issue by leaking the order in which the elements are inserted into the hash table. This was a problem since the Hash function should not be scrutable and no one should know the information. In conclusion, data structures being provided first-class support are less likely to happen because of its rigidness. Generally programming languages that do not have them built-in are more flexible and less prone to issues and can be designed in different ways for them to be more efficient in different applications.