**Mini JSON Parser Prototype Programming Assignment**

**Note: The Visual Studio solution provided must be built in Debug/x86 mode.**

**Background**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. A single or an ordered list of values (in the case of an array). In most languages, the ordered list of values is realized as an array, vector, list, or sequence.

Assume we have the following JSON stored in a C string (string json):

{

"hello": "world",

"t": true,

"f": false,

"n": null,

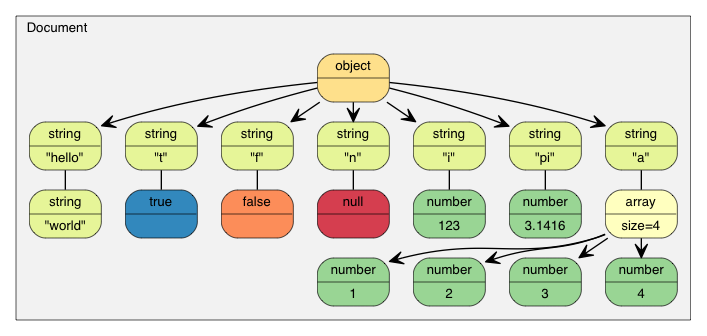
"i": 123,

"pi": 3.1416,

"a": [1, 2, 3, 4]

}

A JSON parser will typically store the data as a document tree.



**Problem**

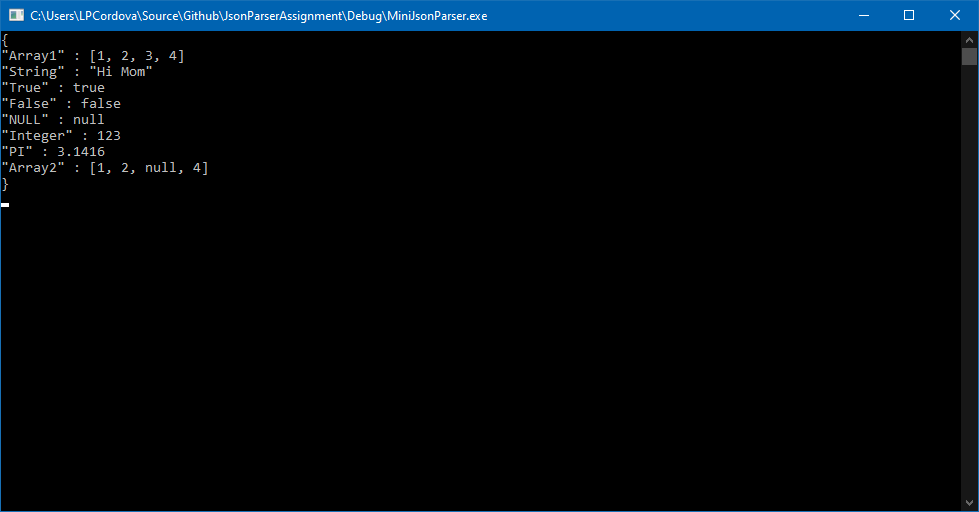
Your task is to build a JSON document tree object to represent the key/value pairs of the JSON string. You are provided with an IJsonDocument interface and a skeleton JsonDocument class. Although you must implement the interface, internally you can use any data structures (from STL or your own) to represent the document that you would like, as long as you adhere to the stipulations.

For your convenience, you are provided a static function called JsonStringExtensions::GetKeyValuePairs(string). The function takes a valid JSON string as a parameter and returns a vector<pair<string, string>> of the key value pairs from the JSON string.

**Example GetKeyValuePairs(string) usage:**

1. static string sample\_json\_ =
2. R"({
3. "A1": [1, 2, 3, 4],
4. "Hi": "Mom",
5. "T": true,
6. "F": false,
7. "NULL": null,
8. "I": 123,
9. "PI": 3.1416,
10. "A2": [1, 2, null, 4]
11. })";
13. vector<pair<string, string>> key\_value\_pairs = JsonStringExtensions::GetKeyValuePairs(sample\_json\_);
15. for (pair<string, string>& pair : key\_value\_pairs)
16. {
17. cout << pair.first << " : " << pair.second << endl;
18. }

**Output**:



**Stipulations**

1. Your internal structure must store each piece of value data as the C++ data type equivalent. The key may be stored as a string. The value must be stored using the C++ data type representing the type. E.g. if you’re representing a Boolean, you must store the value as a bool, if you’re representing an integer, you must store the value as an int, etc.
2. You must implement the IJsonDocument interface. This includes a ToString() function which should output a properly formatted JSON string.
3. The value null represents the lack of a value at any given time. Every supported JSON document type must support the notion of null. For values that are null, your ToString() function must print out *null* as the value.
4. For arrays, assume that the underlying data structure is an array of *<T>*, where T can be any of the types supported by a JSON document tree. Arrays may only contain data of one type but must support that value being null. Note: this does not necessarily mean that you can only solve the problem using C++ templates.
5. You are welcome to use pointers, templates, classes, inheritance or anything you have learned or that you are comfortable with.
6. For this problem, you may use any CPP Reference internet sites or your favorite online references e.g. to look up the syntax for the string find or erase function, etc.
7. You must not leak memory.
8. Exception handling is expected.

**Hints**

1. You may use any method to parse the values into the C++ type. The useful C functions for parsing a C string into a float is atof and for an int, it’s atoi. Of course, there are safer and/or better ways to do these operations, but you are not being measured on that.
2. isdigit(char) will return true if the char parameter is a number.
3. The C++ string type provides a method for converting itself to a C string (const char \*) by using the extension function: .c\_str().

**Deliverables**

1. Commit your code to Github and create a release tag.
2. Submit your URL to Canvas.

**Rubric**

|  |  |
| --- | --- |
| Word or text document explaining your strategy for solving the problem. | 20% |
| Canonical member functions (e.g. constructors, assignment operators, move, etc.) for JsonDocument. | 20% |
| JsonDocument implementation. Insert, Remove, Traverse, ToString. Functional operations. Use of object-oriented principles and C++ features. | 50% |
| Code hygiene. Follows code conventions, clean, concise. Does not leak memory. Throws exceptions when needed. | 10% |
| Total | 100% |