**Requirement For JSON**

Json: JSON is a simple and widely used data-interchange format. It is commonly used for serialising data and sending it over a network, such as in many JavaScript based web applications.

Synatx : Json syntax looks like below.

Json := Element

Element := Value

Value := Object |Array |String |Number |true |false |null

Object := { } |{ Members }

Member := String : Element

Array := [ ] |[ Elements ]

JsonReader and JsonWriter for low level work of parsing and outputting JSON to either file or output stream.

Functional Requirements:

Extracting Data

The parser shall provide functionality to extract data from the parsed JSON structure.

It shall support extraction of primitive data types (strings, numbers, booleans, null) as well as nested structures (objects and arrays).

Error Handling

It shall detect and report errors such as syntax errors, invalid data types, or missing required fields.

Design requirements:

Technical Requirement List to be implemented:

======================================

Data Type:

Write a structure or union which holds all the data types such as primitive, string, objects and arrays.

Write an enum which holds all different token types such a object open "{" , object close "}" , token string, token number etc.

Currently defined simple map to store key and value pair, but in future it will be changed to structure of different data types which also contains map as one of data type.

Reading Data:

Write a Constructor which will read complete Json data from the file. Read the Json data and store it into a string.

To achieve that user need to call reader constructor and pass the file name.

than call readFile method to get the string from the json file. And pass that string to parse method.

Write a Constructor which will read the stream of data and also a string.

Tokenization:

Write a method which will get the current character in the string which need to be parsed. Based on the current character we can write a readToken() method which will be used to Provide token type for further processing string , number, null, true or false. Using this method we can get individual tokens from stream of data.

Write a method skipSpaces() to skip the white spaces, So that we can read proper data and avoid newline, tabs, spaces.

Parser:

Write a parser method which can read list of tokens from string and based on type of token we receive we can parse and find out key and value for each different type using switch case.

Once we receive key and value, It will be inserted into map which is declared of type <std::string, std::string>

Implement parseObject, parseNumber, parseStringString methods to find all the values with respect to key.

Once Basic Version is ready we can change map structure to store more complex values such as object inside an object, list etc. And below is design for the same.

Create a array of pointers which points to map , each map contains an key and a value.

May be we can define as : std::map<std::string, JsonValue> JsonMap;

Note: Currently parseObject is developed in token.cpp file, will be moved into different file in full implimentation.

JsonValue may look like :

std::map<std::string, JsonValue> json\_map;

Acceptance Criteria

1.> empty object

{}

Output :

Empty object

2.>

{

"uuid": "e85e30f8-2922-5fed-a2e9-f5014fb80c38"

}

Output:

cout<<root[uuid].asString();

e85e30f8-2922-5fed-a2e9-f5014fb80c38

3.> Object

{

"uuid": "e85e30f8-2922-5fed-a2e9-f5014fb80c38",

"physicalName": "Eaton 9PX 1500G RT",

"referenceNumber": "9870"

}

Output:

cout << root[uuid].asString() << endl;

cout << root[physicalName].asString() << endl;

cout << root[referenceNumber]<<endl;

e85e30f8-2922-5fed-a2e9-f5014fb80c38

Eaton 9PX 1500G RT

9870

The customer Script:

We open the JSON file (data.json) using an input file stream (std::ifstream).

We read and parse the JSON file into a Json::Value object (root) using the Json::Reader class.

If parsing succeeds, we access and print the parsed JSON data (name, age, city) using the asString() and asInt() functions of Json::Value.

std::Ifstream file("ani.json");

Json::Value root; // to declare the root json object

Json::Reader reader; // To declare the namespace

Reader.parse(file,root);

std::cout << root[name].asString();

std::cout << root[age].asInt();

Example:

{

"uuid": "e85e30f8-2922-5fed-a2e9-f5014fb80c38",

"physicalName": "Eaton 9PX 1500G RT",

"friendlyName": "Eaton 9PX 1500G RT",

"serialNumber": "PA22P46F36",

"partNumber": "9PX1500GRT",

"referenceNumber": "9870",

"vendor": "EATON",

"model": "Eaton 9PX 1500G RT",

"type": "PowerDistribution",

"productName": "Eaton 9PX",

"firmwareVersion": "01.24.6755",

"name": "Eaton 9PX 1500G RT"

}

**Design and implementation details:**

JSON data types include following types.

The simple types are:

Number

String

True

False

Null

The composite types are:

Array - A list of JSON values (Note that array itself is a JSON value so you can have an array of arrays)

Object - It is a set of key-value pairs where you can retrieve any value using its corresponding key.

The actual implementation and data type should be declared something similar to below structure.

classJSONNode{  
 enumclassType{ OBJECT, LIST, STRING, NUMBER, BOOLEAN, NULL\_TYPE};   
 unionValues{  
 JSONObject \*object;  
 JSONList \*list;  
 std::string \*s;  
 floatfValue;  
 boolbValue;  
 } values;  
 Type type;  
}

When we have complicated JSON file such as Json has list of Json objects and array of Json objects which internally have key and values than we can declare each Node recursively as below.

using JSONObject = std::map<std::string, std::shared\_ptr<JSONNode>>;  
using JSONList = std::vector<std::shared\_ptr<JSONNode>>;

For simplicity purpose and based on current requirements we have declared JSON map which stores key and value as below:

std::map<std::string,std::string> json\_map; //This does not supports recursive Json objects.

The Json content should look like :

{

"uuid": "e85e30f8-2922-5fed-a2e9-f5014fb80c38",

"Name": "EOTON",

"Version" : "126.32",

"Value" : "123456",

"Power" : "ALWAYS ON",

"Config": "Start"

}

To return the values wrt json string/file :

Bunch of functions written to parse and handle the Json string/file from Reader class:

bool parse(std::string& document); // read from a document

bool parse(std::istream& is); // read from a stream

bool parse(const char\* beginDoc, const char\* endDoc); // to read the character string after reading from file or input stream

Reader()= default; // constructor

Reader(const std::string& filename);

bool readValue(); // read the data from a json string

std::string readFile(); // read json file to parse

The complete Parsing is divided into three different parts:

1.> Reading from different inputs

2.> Tokenization.

3.> Parsing.

**1.> Reading from different inputs**

User can access Json string using different methods as mentioned below.

1.> calling parse using a document and convert into string and pass to parse.

2.> calling parse using string stream

3.> Calling Parse using a file.

Reader class implements the parse method to parse the input Json string. The input string need to be passed either as a string stream.

Once reading has been done we can use reader[key] to get respected value.

Or else directly pass the file to reader class than call readFile() method to get string than pass it to parse method.

**2.> Tokenization :**

We have to read the Json string and convert it into tokens to process it later and store it into map.

For that, Identify different types of characters that can be read from a Json and tokes will be different types as shown below.

enumclassTOKEN{  
 CURLY\_OPEN,  
 CURLY\_CLOSE,  
 COLON,  
 STRING,  
 NUMBER,  
 ARRAY\_OPEN,  
 ARRAY\_CLOSE,  
 COMMA,  
 BOOLEAN,  
 NULL\_TYPE  
};

In current implementation only required tokes have been included as shown below.

enum TokenType { // enum to store list of different character types while reading json data

tokenEndOfStream = 0,

tokenObjectBegin,

tokenObjectEnd,

tokenString,

tokenMemberSeparator,

tokenError

};

List of utility functions available for supporting tokenization are:

auto readToken(); // find out token from current data

void skipSpaces(); // skip spaces inbetween characters in a json data

bool readString(); // To read the string once encounter " in the readvalue.

bool readObject(Token& token); // to read object

void skipUntilSpace(); // skip spaces

char getNextChar(); // get current character in the stream

bool hasMoreToken(); // to understand if we can create more tokens

**3.> Parsing:**

readValue() method will traverse the string and pick a logically valid token.

Based on the token type different methods will be called for example if the token type is object it will call parseObject() method and from that location onwards it will traverse whole string and push the key and values into JSON map.

List of related methods for parsing:

void parseObject(); // to parse json object

void getMemberNames(); // to iterate though map and display key and value list

std::string& operator[](const std::string&); // overloading to return value wrt key in the map.

List of Implementation pending details:

1.> Json Map structure should be defined to support and store complicated recursive details.

2.> Code should be written to support recursive approach for complex Json data.