CIS*2750
Assignment 2, Module 3

The functions in this module will provide the interface between our parser API, which is written in C, and the web-based GUI in A3 and A4, which will rely on JavaScript and HTML. The different components will communicate using strings in JavaScript Object Notation (JSON) format. We will discuss the JSON format in more detail in a later class. For now, the output format will be provided for you.

These functions will allow your A3 to create and modify Calendar objects from a Web GUI, and save these new objects to disk. Most of these functions are designed to convert the Calendar and its components into JSON strings - and vice versa. We also have a function for adding an Event to a Calendar. You will expand on these functions in A3.

The function descriptions in this document are quite long, for the sake of clarity and precision. However, most the functions themselves will be fairly short and simple, and many will closely resemble the various print... functions you have implemented in Assignment 1.

Please pay careful attention to quotes, spaces, and other details in the output functions. They are important.

New functions

```
char* dtToJSON(DateTime prop);
char* eventToJSON(const Event* event);
char* eventListToJSON(const List* eventList);
char* calendarToJSON(const Calendar* cal);

Calendar* JSONtoCalendar(const char* str);

Event* JSONtoEvent(const char* str);

void addEvent(Calendar* cal, Event* toBeAdded);
```

Description

1. *dtToJSON*

```
char* dtToJSON(DateTime dt);
```

Converts a `DateTime` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"date":"date val","time":"time val","isUTC":utcVal}
```

where `utcVal` are either `true` or `false`. Notice that `utcVal` does not have quotes around it.

For example:
- given a UTC DateTime with the date `19540203` and the time `123012`, we would create the following output string:
  `{"date":"19540203","time":"123012","isUTC":true}`
- given a non-UTC DateTime with the date `19540203` and the time `123012`, we would create the following output string:
  `{"date":"19540203","time":"123012","isUTC":false}`

The format **must** be exactly as specified. Do not add any spaces, newlines, or change capitalization.

The returned string contents for this function - and all the other `...ToJSON` functions below - will contain double-quote characters, so you will need to use the escape sequence `\"`.

This function must not modify its argument in any way.

2. *eventToJSON*

```
char* eventToJSON(const Event* event);
```

Converts an Event struct to a string in JSON format. You can - and should - use `dtToJSON` function defined above.

The function `eventToJSON` must return a newly allocated string in the following format:

```
{"startDT":DTval,"numProps":propVal,"numAlarms":almVal,"summary":"sumVal"}
```

where:

- `DTval` is a JSON value of the Event's `startDateTime` property. This string is returned by `dtToJSON`. Notice that the value does not have quotes around it.
- `propVal` is the total number of properties, i.e. the 3 required properties + the number of properties in `Event->properties` list. This value must always be 3 or more, since the thee required property fields in the Event struct must always have proper values. Notice that the value does not have quotes around it.
- `almVal` is the total number of alarms in `Event->alarms` list. Since alarms are optional, this value might be zero. Again, notice that the value does not have quotes around it.
- `sumVal` is the value of the SUMMARY property in the `Event->properties` list. If this property is not in the list (or list is empty), the value is `""`.

Examples:

1. Simplest event:
- no optional properties, no alarms
- no SUMMARY property
- `startDateTime` is a UTC DateTime with the date `19540203` and the time `123012`

Output:
```
{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":3,"numAlarms":0,"summary":""}
```

2. Event with:
- 1 optional property and 2 alarms
- the SUMMARY property
- `startDateTime` is a non-UTC DateTime with the date `20190211` and the time `143012`

Output:
```
{"startDT":{"date":"20190211","time":"143012","isUTC":false},"numProps":4,"numAlarms":2,"summary":"Do taxes"}
```

3. Event with:
- 3 optional properties and no alarms
- no SUMMARY property
- `startDateTime` is a UTC DateTime with the date `19540203` and the time `123012`

Output:
```
{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":6,"numAlarms":0,"summary":""}
```

The format **must** be exactly as specified. Do not add any spaces, newlines, or change capitalization.

This function must not modify its argument in any way.

If the argument `event` is NULL, the function must return the string `{}` (there is no space there - just two chars).

3. eventListToJSON

```
char* eventListToJSON(const List* eventList);
```

This function will convert a list of Events - i.e. the `events` list of a Calendar - into a JSON string. You can - and should - use `eventToJSON` function defined above.

The function `eventListToJSON` must return a newly allocated string in the following format:

```
[EvtString1,EvtString2,…,EvtStringN]
```

where every `EvtString` is the JSON string returned by `eventToJSON`, and `N` is the number of events in the original list. The order of `EvtString`'s must be the same as the order of events in the original list.

For example, assume that the argument `eventList` is a list containing two events:

- First one with no optional properties, no alarms, and no SUMMARY
  - `startDateTime` is a UTC DateTime with the date `19540203` and the time `123012`
- Second one with 1 optional property, 2 alarms, and a SUMMARY with the value `Do taxes`
  - `startDateTime` is a UTC DateTime with the date `19540203` and the time `123012`

The function would then allocate and return the string:

```
[{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":3,"numAlarms":
0,"summary":""},{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":
4,"numAlarms":2,"summary":"Do taxes"}]
```

Please note that the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no spaces, and look like this (sorry for the teeny font):

```
[{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":3,"numAlarms":0,"summary":""},{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":4,"numAlarms":2,"summary":"Do taxes"}]
```

If the argument `eventList` is a list containing one event, e.g. :

- No optional properties, no alarms, and no SUMMARY
- `startDateTime` is a UTC DateTime with the date `19540203` and the time `123012`

the function would then allocate and return the string (notice no comma before `{` or after `}`):

```
[{"startDT":{"date":"19540203","time":"123012","isUTC":true},"numProps":3,"numAlarms":0,"summary":""}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `eventList` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

4. calendarToJSON

```
char* calendarToJSON(const Calendar* cal);
```

Converts a Calendar struct to a string in JSON format. The function must return a newly allocated string in the following format:

`{"version":verVal,"prodID":"prodIDVal","numProps":propVal,"numEvents":evtVal}`

where
- `verVal` is the version
- `prodIDVal` is the product ID
- `propVal` is the total number of properties, i.e. the 2 required properties + the number of properties in `Calendar->properties` list. This value must always be 2 or more, since the two required property fields in the Calendar struct must always have proper values.
- `numEvents` is the total number of events, i.e. the number of events in `Calendar->events` list. This value must always be 1 or more, since the event list must never be empty.

Examples:

- Simplest calendar: no optional properties, one event (based on testCalSimple file posted for A1 on Moodle):
  `{"version":2,"prodID":"-//hacksw/handcal//NONSGML v1.0//EN","numProps":2,"numEvents":1}`
- Calendar with 2 optional properties, two events:
  `{"version":2,"prodID":"-//hacksw/handcal//NONSGML v1.0//EN","numProps":4,"numEvents":2}`

The format **must** be exactly as specified. Do not add any spaces, newlines, or change capitalization. As always, pay close attention to the quotes.

This function must not modify its argument in any way.

If the argument `event` is NULL, the function must return the string `{}` (there is no space there - just two chars).

5. JSONtoCalendar

`Calendar* JSONtoCalendar(const char* str);`

This function create a very simple `Calendar` object from a JSON string. The Calendar will be partially incomplete; you'll fill in the missing pieces in Assignment 3.

The new `Calendar` object will have a valid `version` and `prodID`. The `properties` list must be initialized, but empty. The `events` list must also be initialized, but empty. Yes, this will create an invalid calendar, but that's OK for now. I will give you some flexibility on how to fix this in Assignment 3, which is why we are leaving it empty.

The format is similar to the JSON string created by `calendarToJSON`, but without the list info:
`{"version":verVal,"prodID":"prodIDVal"}`

For example, given the string:
`{"version":2,"prodID":"-//hacksw/handcal//NONSGML v1.0//EN"}`

`JSONtoCalendar` would create a new `Calendar`:
- `version` field is set to 2
- value of the `prodID` field is the string `-//hacksw/handcal//NONSGML v1.0//EN`
- the `events` list will be empty (i.e. initialized, length 0).
- the `properties` list will be empty (i.e. initialized, length 0).

The function must allocate, initialize, and return the new `Calendar` struct.

This function must not modify its argument in any way.

If the input string `str` is NULL, return NULL.

If the input string `str` cannot be parsed correctly for any reason, return NULL.

6. *JSONtoEvent*

```
Event* JSONtoEvent(const char* str);
```

This function create a very simple `Event` object from a JSON string. The Event will be partially incomplete; you will fill in the missing pieces in Assignment 3.

The new `Event` object will have a valid `UID`. The `creationDateTime` and `startDateTime` fields will be left uninitialized. The `properties` and `alarms` lists must be initialized, but empty. Again, technically this is not yet a valid Event, but I will give you some flexibility on how to set the dates in Assignment 3, which is why we are laving them empty for now.

The input string will be in the following format:

```
{"UID":"value"}
```

For example, given the string:
`{"UID":"1234"}`

`JSONtoEvent` would create new event:
- the UID field will contain the string `1234`
- The `properties` list will be empty (i.e. initialized, length 0)
- The `alarms` list will be empty (i.e. initialized, length 0)
- The `creationDateTime` and `startDateTime` fields will be left uninitialized.

The function must return the new `Event`.

This function must not modify its argument in any way.

If the input string `str` is NULL, return NULL.

If the input string `str` cannot be parsed correctly for any reason, return NULL.

7. *addEvent*

```
void addEvent(Calendar* cal, Event* toBeAdded);
```

This function adds a Property struct to the Card object by inserting it at the end of the `events` list.

It must not modify the argument `toBeAdded` in any way.

If either of the arguments is NULL, the function must do nothing.