CIS*2750
Assignment 1
Deadline: Monday, February 4, 9:00am
Weight: 15%

Module 1: Primary functions

This is Module 1 of Assignment 1. Module 2 will be much smaller, and will deal specifically with error values and error handling. It will be released after January 21. Module 2 may also contain additional submission details.

Description

In this assignment, you need to implement a library to parse the iCalendar files. The link to the format description is posted in the Assignment 1 description. Make sure you understand the format before doing the assignment.

According to the specification (RFC 5545), an iCalendar object contains:
- two specific required properties that must appear exactly once
- several optional properties
- one or more component of several different types

Our Assignment 1 parser will assume a somewhat simpler iCalendar file:
- two specific required properties (see the format description)
- multiple optional calendar properties
- multiple event components, which may contain alarms

This structure is represented by the `Calendar` type in `CalendarParser.h`.

According to the specification (RFC 5545), an event component contains:
- two specific required properties that must appear exactly once
- an arbitrary number of optional properties
- 0 or more alarm components
- Note: Since the METHOD property of the Calendar is optional, we will make the DTSTART property required for an our implementation of the event. Therefore we have the `startDateTime` property in the `Event` struct.

Similarly, an alarm component also contains two required properties and an arbitrary number of optional ones.

These structures are represented by the `Event` and `Alarm` types in `CalendarParser.h`

The `Property` struct represents a generic property. The `propName` contains the property name (e.g. ORGANIZER). The `propDescr` contains everything else - i.e. property value(s) and the parameters (if parameters are present). Make sure that you do not include the delimiter separating the property name from the parameter list or values. For example:
- If the property line contains STATUS:CONFIRMED
  `propName` is `STATUS` and `propDescr` is `CONFIRMED`

- If the property line contains ORGANIZER;CN=Some Dude;mailto:some_dude@mywebmail.com
  `propName` is `ORGANIZER` and `propDescr` is `CN=Some Dude;mailto:some_dude@mywebmail.com`

The header also contains a dedicated type for representing the date-time properties DTSTART and DTSTAMP, a few headers, and a couple of other helpful definitions.

Note that the parser **must** handle folded lines exactly as described in the iCalendar specification.

Your assignment will be graded using an automated test suite, so you must follow all requirements exactly, or you will lose marks.

Required Functions

Read the comments in `CalendarParser.h` carefully. They provide additional implementation details. You **must** implement **every** function in `CalendarParser.h`; they are also listed below. If you do not complete any of the functions, you **must** provide a stub for every one them.

Applications using the parser library will include `CalendarParser.h` in their main program. The `CalendarParser.h` header has been provided for you. Do not change it in any way. `CalendarParser.h` is the public "face" of our calendar API. All the helper functions are internal implementation details, and should not be publicly/globally visible. When we grade your code, we will use the standard `CalendarParser.h` to compile and run it.

If you create additional header files, include them in the `.c` files that use them.

Calendar parser functions

```
ICalErrorCode createCalendar(char* fileName, Calendar** obj);
```

This function does the parsing and allocated a Calendar object. It accepts a filename and an address of a Calendar pointer (i.e. a double pointer). If the file has been parsed successfully, the calendar object is allocated and the information is stored in it. The function then returns `OK`.

However, the parsing can fail for various reasons. In that case, the obj argument is set to `NULL` and `createCalendar` returns a variety of error codes to indicate this.

What `ICalErrorCode` value should be returned in what situation will be discussed in Module 2. For now, you can return `OK` if the file is successfully parsed, and `INV_FILE` otherwise.

```
char* printCalendar(const Calendar* obj);
```

This function returns a humanly readable string representation of the entire calendar object. It will be used mostly by you, for debugging your parser. It must not modify the calendar object in any way. The function must allocate the string dynamically.

```
void deleteCalendar(Calendar* obj)
```

This function deallocates the frees object, including all of its subcomponents.

```
char* printError(ICalErrorCode err)
```

This function returns a string based on the `ICalErrorCode` value to make the output of your program easier to understand - i.e. "OK" if err is `OK`, "INV_FILE" or "invalid file" is `INV_FILE` was passed, etc.. The function must allocate the string dynamically. For now, you can return `OK` if the file is successfully parsed, and `INV_FILE` otherwise; additional details will be provided in Module 2.

Helper functions

In addition the above functions, you must also write a number of helper functions. We will need to store the types `Event`, `Alarm`, and `Property` in lists. We will also need to print and delete `DateTime` values, and may need to compare them in future assignments:

```
void deleteEvent(void* toBeDeleted);
int compareEvents(const void* first, const void* second);
char* printEvent(void* toBePrinted);
```

```
void deleteAlarm(void* toBeDeleted);
int compareAlarms(const void* first, const void* second);
char* printAlarm(void* toBePrinted);


void deleteProperty(void* toBeDeleted);
int compareProperties(const void* first, const void* second);
char* printProperty(void* toBePrinted);


void deleteDate(void* toBeDeleted);
int compareDates(const void* first, const void* second);
char* printDate(void* toBePrinted);
```

- this function can be a stub for now, e.g. always return 0. We will flesh it out later, if/when we need it.

Additional guidelines and requirements

In addition, it is strongly recommended that you write additional helpers functions for parsing the file - e.g. parsing a property, parameter, or a date-time. You should also write delete...() functions for all the structs, since they will all be dynamically allocated.

All required functions **must** be in a file called `CalendarParser.h`. You are free to create your additional "helper functions" in a separate `.c` file, if you find some recurring processing steps that you would like to factor out into a single place. Do **not** place headers for these additional helper function in `CalendarParser.h`. They must be in a separate header file, since they are internal to your implementation and not for public users of the utility package.

For your own test purposes, you will also want to code a main program in another `.c` file that calls your functions with a variety of test cases> However, you **must not** submit that program. Also, do **not** put your main() function in `CalendarParser.h`. Failure to do this will cause the test program will fail due to multiple definitions of main(); **you will lose marks for that**, and may get a 0 for the assignment.

Your functions are supposed to be robust. They will be tested with various kinds of invalid data and must detect problems without crashing. If your functions encounter a problem, they must free all memory and return.

Function naming

You are welcome to name your helper functions as you see fit. However, **do not** put the underscore (_) character at the start of your function names. That is reserved solely for the test harness functions. Failure to do so may result in compiler errors due to name collisions - and a grade of zero (0) as a result.

Linked list

You are expected to use a linked list for storing various calendar components. You can use the list implementation that I will use in the class examples in Week 2. You can also use your own. However, your implementation **must** be compliant with the List API defined in `LinkedListAPI.h`. Failure to do so may result in a grade deductions, up to and including a grade of zero.

(Strongly) Recommended development path:

1. Implement a simple parser that extracts the required properties of the Calendar (version and product ID).
2. Add a basic `deleteCalendar` functionality and test for memory leaks
3. Add a basic `printCalendar` functionality and test for memory leaks

4. Add handling of multiple optional properties
    1. Update `deleteCalendar` and `printCalendar` functionality.
    2. Test for memory leaks
5. Add line unfolding. Line unfolding is part of the format specification and you **must** implement it to receive full marks.
    1. Test for memory leaks
6. Add handling of minimal Events.
    1. Update `deleteCalendar` and `printCalendar` functionality.
    2. Test for memory leaks
7. Add handling of Events with properties
    1. Update `deleteCalendar` and `printCalendar` functionality.
    2. Test for memory leaks
8. Add handling of minimal Alarms
    1. Update `deleteCalendar` and `printCalendar` functionality.
    2. Test for memory leaks
9. Add handling of Alarms with properties
    1. Update `deleteCalendar` and `printCalendar` functionality.
    2. Test for memory leaks
10. Implement proper error code returns (Module 2)
11. Add `printError` functionality
12. Test for memory leaks

Important points

Do:

- **Do** be careful about upper/lower case.
- **Do** include comments with your name and student ID at the top of every file you submit

Do not:

- **Do not** submit `CalendarParser.h` or `LinkedListAPI.h`. If they are submitted, they will be overwritten.
- **Do not** change the given typedefs or function prototypes `CalendarParser.h`
- **Do not** hardcode any path or directory information into #include statements, e.g. `#include "../include/SomeHeader.h`
- **Do not** submit any `main()` functions
- **Do not** use `exit()` function calls anywhere in your code. Since you're writing a library, it must always return the control to the caller using the `return` statement
- **Do not** exit the program from one of the parser functions if a problem is encountered, return an error value instead.
- **Do not** print anything to the command line.
- **Do not** assume that your pointers are valid. Always check for NULL pointers in function arguments.

Failure to follow any of the above points may result in loss of marks, or even a zero for the assignment if they cause compiler errors with the test harness.

Submission structure

The submission must have the following directory structure:

`assign1/`	- contains the `Makefile` file.
`assign1/bin`	- should be empty, but this is where the `Makefile` will place the shared lib files.
`assign1/src`	- contains `CalendarParser.c` , `LinkedListAPI.c`, and your additional source files.
`assign1/include`	- contains your additional headers. Do not submit `CalendarParser.h` and `LinkedListAPI.h`.

Makefile

You will need to provide a `Makefile` with the following functionality:
- `make list` creates a shared library `liblist.so` in `assign1/bin`
- `make parser` creates a shared library `libcal.so` in `assign1/bin`
- `make` or `make all` creates `liblist.so` and `libcal.so` in `assign1/bin`
- `make clean` removes all `.o` and `.so` files

Evaluation

Your code must compile, run, and have all of the specified functionality implemented. Any compiler errors will result in the automatic grade of **zero** for the assignment. Infinite loops will also result in a grade of **zero**.

Marks will be deducted for:
- Incorrect and missing functionality
- Deviations from the requirements
- Run-time errors, including infinite loops
- Compiler warnings
- Memory leaks reported by valgrind
- Memory errors reported by valgrind
- Failure to follow submission instructions

Submission

Submit your files as a Zip archive using Moodle. File name must be `A1FirstnameLastname.zip`.

Late submissions: see course outline for late submission policies.

This assignment is individual work and is subject to the University Academic Misconduct Policy. See course outline for details)