CIS*2750
Assignment 1
Deadline: Monday, February 10, 11:59pm
Weight: 16.6%

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 in about a week.

Description

Our overall project, discussed in Lecture 1, will span Assignments 1 - 3. It will be an app written in C and Python that will create class lists from vCard files and do some basic class and grade management.

In this assignment, you will implement a the first component of this app - a library to parse vCard files. The link to the format description has been posted in the Assignment 1 description. Make sure you understand the format before doing the assignment.

According to the specification (RFC 6350), a vCard object contains:
- one specific required property that must appear at least once
- multiple optional properties, which may contain parameters

Our Assignment 1 parser will assume a somewhat simpler vCard file:
- You can always assume that the parameters of a property (if any) will NOT contain the ":" character. In other words, you can rely on having the ":" character separate the value(s) of a property from everything else

This structure is represented by the `Card` type in `VCParser.h`.

According to the specification (RFC 6350), a property component contains:
- 1 or more values
- 0 or more parameters

These elements are represented by the `Property` and `Parameter` types in `VCParser.h`. Property values are strings.

The header file `VCParser.h` also contains a type for representing the date-time property, a couple of helpful definitions, and the various standard C headers you will need.

Your assignment will be graded using an automated test suite, so you must follow all requirements exactly, or you will lose marks - and possibly get a zero on the assignment.

Required Functions

Read the comments in `VCParser.h` carefully. They provide additional implementation details and are an important part of the documentation for this assignment. You **must** implement **every** function in `VCParser.h`; they are also listed below. If you cannot complete any of the functions, you **must** provide a stub for every one them.

Applications using the parser library will include `VCParser.h` in their main program. The `VCParser.h` header has been provided for you. Do not change it in any way. `VCParser.h` is the public “face” of our card 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 `VCParser.h` to compile and run it.

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

Card parser functions

```
VCardErrorCode createCard(char* fileName, Card** newCardObject);
```

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

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

Regarding file extensions - both `.vcf` and `.vcard` are stated in the specification, so both must be accepted by the parser. All other file extensions must not be accepted.

```
char* cardToString(const Card* obj);
```

This function returns a humanly readable string representation of the entire card object. It must not modify the card object in any way. The function must allocate the string dynamically. The format of the output string is up to you - it will be useful to you when debugging your own code.

```
void deleteCard(Card* obj);
```

This function deallocates the `Card` object - i.e. frees all memory associated with it, including all of its subcomponents.

```
const char* errorToString(VCardErrorCode err);
```

This function is meant to make the error codes more humanly readable. It returns a string based on the `VCardErrorCode` 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. 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 `Property`, `Value`, and `Parameter` in a list. We will also need to print and delete `DateTime` values, and may need to compare them in future assignments:

```
void deleteProperty(void* toBeDeleted);
int compareProperties(const void* first, const void* second);
char* propertyToString(void* prop);
```

```
void deleteParameter(void* toBeDeleted);
int compareParameters(const void* first, const void* second);
char* parameterToString(void* param);

void deleteValue(void* toBeDeleted);
int compareValues(const void* first, const void* second);
char* valueToString(void* val);

void deleteDate(void* toBeDeleted);
int compareDates(const void* first, const void* second);
char* dateToString(void* date);
```
- this function can be a stub for now, e.g. always return 0. We will flesh it out later, when we need it.

Additional guidelines and requirements

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 files prefaced with `VC` - e.g. `VCParser.c`, `VCHelpers.c`, etc.. 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 `VCParser.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. This separate header file also **must** be prefaced with `VC`, e.g. `VCHelpers.c`.

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, but you won't submit that program. The file containing the main function **must not** have the `VC` prefix.

Do **not** put your main() function into any of the files with the `VC` prefix. Otherwise, the test executable will fail due to multiple definitions of main(); **you will lose marks for that**, and may get a zero 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 run-time 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 vCard components. You can use the list implementation that I use in the class examples. 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.

Recommended development path:

1. Start by implementing a simple parser that extracts the FN property
2. Add a basic `deleteCard` functionality and test for memory leaks
3. Add a basic `cardToString` functionality and test for memory leaks
4. Add handling of multiple optional properties with single values
   1. Update `deleteCard` and `cardToString` functions
   2. Test for memory leaks
5. Add handling of properties with compound values
   1. Update `deleteCard` and `cardToString` functions
   2. Test for memory leaks
6. Add line unfolding. Line unfolding is part of the format specification and you **must** implement it to receive full marks.
   1. you guessed it
   2. you guessed it
7. Add handling of properties with parameters
   1. Have a beer, and ignore the rest of the assignment.
   2. Just kidding. Yes, keep updating `delete`/`toString` functions, and testing for leaks
8. Implement proper error code returns (Module 2)
9. Add `errorToString` functionality
10. 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** use the `VC` prefix for all .c and .h files that you wan to include in your graded A1

Do not:

- **Do not** change the given typedefs or function prototypes `VCParser.h`
- **Do not** hardcode any paths or directory information into #include statements, e.g. `#include "../include/SomeHeader.h`
- **Do not** put the `main()` function into any of the files with the `VC` prefix.
- **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. Note: be careful if you use Git, since it can ignore empty directories, and this can mess up your submission
`assign1/src`	- contains `VCParser.c` , `LinkedListAPI.c`, and your additional source files.
`assign1/include`	- contains your additional headers. Do not submit `VCParser.h` and `LinkedListAPI.h`. If you do, they will be deleted and replaced with the standard ones that are posted on the course website.

Makefile

You will need to provide a Makefile with the following functionality:
- `make parser` creates a shared library `libvcparser.so` in `assign1/bin`
- `make clean` removes all `.o` and `.so` files
- You are welcome to add additional targets for your own purposes, e.g. testing. We will not run them.

Evaluation

Your code will be tested by an automated harness. You will submit a Makefile, which will be used to produce a shared library. This library will then be tested on the standard CIS*2750 Docker images by the TAs, who will use a precompiled executable file containing the test harness, as well as another executable file with simple memory leak tests. Your library must implement the assignment API exactly as specified, or you will get run-time errors because the executable files will not find functions in the library that they expect.

Your code must compile, run, and have all of the specified functionality implemented. Any compiler errors will result in the automatic grade of **zero (0)** for the assignment. Infinite loops will also result in a grade of **zero (0)**.

Marks will be deducted for:
- Incorrect and missing functionality
- Deviations from the assignment requirements
- Run-time errors, including infinite loops
- Compiler warnings
- Memory leaks reported by valgrind
- Memory errors reported by valgrind
- Failure to follow submission instructions, e.g. incorrect archive type, incorrect naming of targets, etc..

Submission

Submit your files - source code and Makefile, as described above - as a Zip archive using Moodle. File name must be `A1FirstnameLastname.zip`. Please do not use any other archive formats, or you will lose marks.

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)