Ethan Simmonds (1423402)

DEVELOPER ASSESSMENT

REPORT

Dear Junior Developer,

Unacceptable! Much of this work is painfully amateurish and I would expect better from someone working at this company - even from a junior developer position. To ensure our company mantra of developing high-quality solutions is drummed into each and every employee, I will offer some sage wisdom and advice to you in the hopes the output quality of your work is much better in the future.

Code Style

Multiple incoherencies in your coding style and convention makes it so nobody but you is able to understand what is happening in your program. Incorrect bracket usage coupled with poor naming choice for your functions - which I may include lack ANY comments - fail to quantifiably describe what they are doing at immediate glance. My advice for you is to consider the programming layman when writing code. The lack of generalization will drive any formidable developer crazy!

On the mention of comments, you aren't being asked to explain every line. Very top-level descriptions above every function header will suffice for this task. Please check your spelling is correct as well (there is a particularly egregious case if you take a slight glance).

Resource checking

There are a handful of examples demonstrating negligence of your resources and how they are used. Two large and important examples which indicate inexperience or poor fundamentals of the C programming language is files never being closed as well as memory never being deallocated after use. In the quest of developing high-quality solutions it is crucial programs start and end cleanly using minimal resources as possible. I'm sure you don't need to be explained why these are important concepts to a programmer. They have been remedied in the follow-up submission I have given you.

Many language-specific errors would likely surface beforehand if you chose to have them display during compilation. Your makefile currently does not. I suggest you change that.

Implementation process

I suggest you reconsider the implementation of how the program is finding clients. In your submission you chose to sort clients by attribute before searching. This is $O(n^2)$ time complexity; very unsuitable especially when it can be simplified to a linear search of O(n) (worst case) complexity. The necessity of sorting before searching will not be required on such a small data set (fifty clients as shown on line 98) and is wholly discouraged for this scale of task. For the sake of your understanding I have remedied the errors in your sort functions however do realise they are not needed at all.

I advise you mull over these points and adjust your practices accordingly. Attached at the bottom is a set of changes made to your program.

User Interface

The User Interface you've given is very threadbare and doesn't fully convey what the user is supposed to be doing. It is very confusing to operate since it doesn't return any sort of prompt. It would be nice to know when the program has actually started so I know it is ready to interact with - even if it goes straight to the input menu. Right now it appears to be hanging when starting but really it is just awaiting input. My first thought was that something had gone wrong and it was broken. Certainly wasn't a great first impression! If a developer with a technical background doesn't understand what is going on you can be sure the average unassuming user will be equally (if not even more) confused. I will reiterate what I said in another section that you should consider the programming layman and whether they can fully navigate the program without any sort of external intervention. Keep definitions and language simple so that everyone can understand. If you could solve that issue then developers will have no problem doing the exact same operation.

Feedback is a very important part of the user experience and should follow some sort of convention or template - even if it is designed yourself. The main practices when telling users about the status of the program is to:

- Distinguish the output from the input. Line breaking is a simple and effective method.

Ethan Simmonds (1423402)

- Map each action with an appropriate response. Does the user want to search phone numbers? It would make logical sense for the program to return some sort of confirmation it is doing exactly this and not something else.
- Technical jargon in the interface must be as minimal as possible.
- When applicable, updating the user when the program is performing tasks.
- When menuing, place required input before any descriptions. (*0 -> exit* is an example)

Also, please make sure the interface presentation is sharp and correct. Watch the formatting of your print statements already in the program.

Kind Regards,
Ethan

Changes

- struct *s* renamed to struct *client.*
- References to *emialAddress* changed to *email.*
- Struct field names changed.
- Duplicate inclusion of stdio.h has been removed/fixed.
- Booleans used for search functions.
- Pointer formatting fixed.
- Function header names much more clearer.
- For loop fixes (such as variable *i* missing in sph).
- Static integer variables *i*, *j* no longer global - referenced within functions instead.
- Search functions have their *while* statements turned into *for* statements.
- Search functions now use *strcmp.*
- Function *freeClients* added to deallocate memory from clients + struct.
- Bracketing fixes (mainly in loops).
- *FILE f* changed to *FILE file*. For consistency's sake.
- Input checking from file is fixed.
- File now successfully closes after usage.
- *Math.h* inclusion removed (not needed).
- Most print statements in the main function block are much more descriptive.
- Memory allocation now uses *sizeof(char)* instead of *sizeof(val[0)).*
    - A handful of similar examples were fixed too.
- Program breaks out of execution properly when command is *0* (memory properly freed too).
- Functions which return integers now return booleans (not an issue, but for coherency).

Ethan Simmonds (1423402)