E14 – Bug Squashing

# Report

Dear Junior Developer.

I have read, edited, and returned your code for you. Unfortunately, I am not particularly impressed with your efforts; I would expect a high school student to have more concise and understandable code than this. The largest areas of weakness in your program were:

1. Inconsistent programming style
2. Lack of comments
3. Unclear and ambiguous methods
4. Incorrect use of the C language
5. Inefficient approach to the problem
6. Inconsistent programming style

My best advice would be to pick one way that you like coding and *stick with it.* Your code had many small syntax differences that made it difficult to follow – simply because of the inconsistency. For example, if you use curly brackets on one if statement, use them on all. If you declare a pointer with no spaces between the asterisk and the type, do this for all. If you indent within a while loop, indent within them all. You get the idea. Also, always create the close curly brace as you create the open. There were sections within the code that did not close off when they were intended. This makes it difficult for another party having to read your program.

1. Lack of comments

No code submitted or shared in a working environment (or any, truthfully) should be without comments. It may seem a tedious task, but it is essential in understanding someone else’s work. Without comments, it is ambiguous and sometimes even impossible to understand what is meant to be going on. I have left comments above your methods with questions that should hopefully prompt you in how to correctly fill these in.

1. Unclear and ambiguous methods

This may follow on from the previous point, but it took significant time to understand your methods. A good programmer would always follow standard and informative naming conventions, only using abbreviations where unambiguous and appropriate. I have renamed your methods and a few of your variable to demonstrate this.

1. Incorrect use of the C language

Your code does not compile without error. The *makefile* provided in fact redirects any errors away from the user and simply hides them. I would not recommend this at all. Use the error messages to debug and improve your code until there are no warning or errors. One of the issues in the program was pointers. Many areas, including *scanf()* were expecting different variable/pointer types than you passed to them. I have since corrected this. Refer to the notes attached to see the full list of corrections.

Other areas of fault were:

* Not closing the file after reading it
* Using ampersands on string input variables
* Memory allocation calls not using the correct type, or in the correct place
* Not freeing any of your memory allocations
* Not returning exit success/failure integers in main
* Using the primitive form *‘=’* to check two strings are the same.

1. Inefficient approach to the problem

The task required was to ‘find people in a list of client data’. Setting up the struct was a great start, however, it appears you are attempting to sort the data on each separate attribute before searching for it. I urge you to think of the inefficiencies in doing this. The sort you have implemented is O(n2), whereas simply searching for the attribute itself without sorting is only O(n) worst case. Implementing a sort and search algorithm such as a Binary Search Tree would be excellent if many searches need to be performed quickly on large sets of data. You will need to consider the input data your program will be working with and proceed accordingly. Another difficulty to consider here is the four separate attributes required to search upon; a typical BST searches on only one. My recommendation is to stick with simple linear search (partly as your program implied you shall never have more than 50 clients at one time - within the for loop reading data in from the file). If this is not the case, reconsider your approach. One last thing I will mention on this, what are you required to do when your search finds a match? Currently the program confirms the find, however, it might be more beneficial to print out all records of that client should there be a match. Give a little thought about how you would go about implementing this.

Although I appreciated your attempt, your program was in a particularly poor state when it arrived on my desk. In future I would recommend refreshing your memory on the use of pointers, memory allocation, and user input in C, as well as testing your code after every few lines written. This helps narrow down where a bug has been introduced. Best of luck with your future programming endeavours.

Kind regards,

Nikolah Pearce

P.S. Please see attached a full list of the notes created while correcting your program.

# Corrected parts

* Renamed file *client.c*
* Renamed the struct from S to *client*
* Removed duplicate *import <std.io>* declaration
* Added comments to all methods
* Changed the compile method to remove the redirection of compile errors
* Created a README.txt file to show clearly how to use the program
* Expanded method names to be more informative
* Renamed non-counting variables to be more informative
* Corrected variable name ‘*emialAddress’* to correct spelling: *emailAddress*
* Improved grammar and punctuation in print statements
* Added braces to all if statements and while loops
* Corrected indentation on the switch case and other areas such as *while()* loops
* Removed static declarations of *i* and *j* integers, put these in the methods that used them.
* Removed multiple declaration of variable count within main (as we have count already set to a static variable)
* Set integers *i* and *j* to be 0 in for loops
* Added print statements to inform the user how to work the program
* Replaced the inefficient *get()* method with *scanf()* for user input
* Removed the ampersand, *&,* from in front of string input variables
* Corrected pointer declarations to be consistent on the placing of the asterisk
* Corrected the ‘swap’ cases in the sort methods – some were missing declaration of the temporary store variable
* Removed the inefficient sort and then linear search process, leaving just the linear search
* Corrected string comparisons by using *strcmp()* instead of *‘=’*
* Removed the incorrect while incrementation and replaced with a for loop
* Changed magic number of 10 (in a *while()* loop) to be more proper (-1)
* Moved memory declarations to be in the correct place/loop
* Corrected memory allocation to use the type as in *sizeof()*
* Freed memory allocated to some internal variables
* Added a *free\_all()* method to free the memory allocated associated with the struct