Prof. Eric Ferguson

Seneca

PRG255 Lab

Ora – C – le

"We ~~can't~~ promise you aren't just a number."

Welcome to your first day at Globecks Corp. programmer! Our database staff have been hard at work getting our employee management system online, but we still don't have a way to interact with it in a way that the rest of us understand. You are tasked for the next month to build an interfacing program that can do this, so our IT administrators can easily and securely modify, add, or delete personal employee data. Good luck!

Background:

In real world applications, a program is built as a tool to make more complicated (typically user) operations easier, while simultaneously being vigilant in providing data security to any sensitive areas. This is typically done by leveraging permissions to more advanced features, either through roles or custom rules.

Let's take any average front end of a web page, and think about how a user accesses their personal data. Users will typically start with an email or username and a password, as this is sufficient (usually) to access their personal data. From an administrator stand point, it is both a vulnerability, and impossibility, to commit all emails and passwords to memory (not to mention a gross breach of user trust). So what do we do? We assign every user a unique identifier which is our backend lookup, that's also strong enough to not be guessed easily by any malicious party. Using this, we begin the work of modifying data.

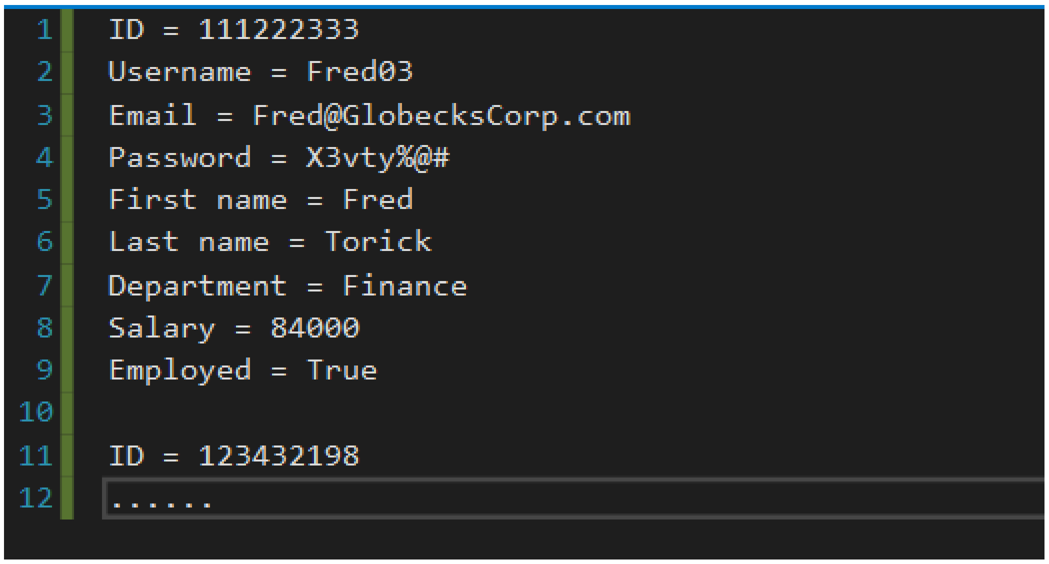
Part 1: Build-A-User (2 Marks)

Let's start by building a user, which is defined by what we will be sending and receiving from the database. The team has given you a list of parameters that line up with their system, which we need to also have. Make a struct called employee, with the below fields.

Char* Username	Char* Email	Char* Password
Char* ID	Char* firstName	Char* lastName
Double Salary	Char* Department	Bool isEmployed

Part 2: Build a Data Store (2 Marks)

In a typical system here is where we would begin tying into our database, but for emulation purposes and without a database software, let's build our own. Inside of the pre-built "Resource Files" folder visible in our solution explorer, add 3 text files. The first file call EmployeeList.txt, the second StagedChanges.txt, and the last Log.txt. The first 2 files will look similar in appearance, with the exception of EmployeeList.txt being populated with 10 pre-defined employees that you will make. StagedChanges.txt can start empty, as we will write into this file when making changes before this is applied to the master list. An example employee is modeled below.



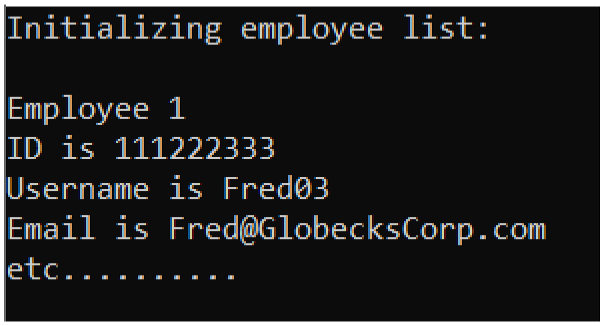
```
ID = 111222333
Username = Fred03
Email = Fred@GlobecksCorp.com
Password = X3vty%@#
First name = Fred
Last name = Torick
Department = Finance
Salary = 84000
Employed = True

ID = 123432198
......
```

Part 3: Generate Employee List Function (3 marks)

Now that we have our data, let's get it processed into our program so we can manipulate it. Let's make this easier, and break it down into a few important steps:

- Make a struct array of employees, with 10 extra empty employee elements.
- Read each field per employee into each variable of a struct array element
- Print to the console a verification of all variables per element from the struct array.



```
Initializing employee list:

Employee 1
ID is 111222333
Username is Fred03
Email is Fred@GlobecksCorp.com
etc..........
```

Part 4: Menu - Add Employee Function (2 Marks)

Before we begin developing features, *build a console interface* that allows the user of the program to either add, delete, or modify an employee record. For add, your program must find the next available employee element, and prompt the user to enter all field information (i.e. username, email etc.). Save this record into the element, and write the revised entire list to the StagingChanges.txt file, as well as append a relevant line to the log file that this has occurred. An example might be “Added employee ID X in position Y in the struct array”.

Part 5: Menu - Delete Employee Function (2 Marks)

Deleting an employee record will follow a similar structure as adding one, with the exception of now first finding a specific array element. To begin, write a lookup function that accepts an ID (provided by the user) and struct array, and returns a reference to array element. The delete will now delete all field data, making the employee record empty. While the program is running, this is treated as a new empty record to the add function. Write the revised entire list to the stagingChanges.txt file, as well as append a relevant line to the log file that this has occurred. An example might be “Removed employee ID X in position Y in the struct array”.

Part 6: Menu - Modify Employee Function (2 Marks)

Using our previously built lookup function, find the employee element that the user has requested from the menu function. The user should be prompted to confirm all previous data is correct, and then make changes to the field of their choice. Save all changes to the element, and write the revised entire list to the stagingChanges.txt file, as well as append a relevant line to the log file that this has occurred. An example might be “Modified employee ID X in position Y in the struct array”.

Part 7: Override List Function (2 Marks)

Now that all changes have been completed, ensure these are written into our permanent storage, or in this case our EmployeeList.txt file.

- Copy the contents of the stagedChanges.txt file over to the EmployeeList.txt file
- Empty the StagedChanges.txt file
- Print the change log of each option chosen and associated ID.
- Print out the final employee list to the console