CYBR 401
Homework 1

OVERVIEW

For this homework assignment, we will be creating a very simple program that will take user input, validate it, and allow a user to add users to a database. While we could add boatloads of extra functions to this program, I want you to be able to get it done in a few weeks. So, we'll keep it pretty limited.

Here's how the program works. Upon boot up, the program prints a *welcome message.* It then indicates that the program is beginning and asks the user to enter a *number* of users that they would like to enter. After entering an *invalid* integer, the program will indicate that the input was wrong, then will ask again for an integer. When a *valid* integer is entered, the program will allow the user to enter the chosen number of users into the database.

Each user has a *user number*, a *username*, a *first name*, and a *last name*. Each iteration will ask the user to input this data. Once this input is entered, the program will then *display* that it is beginning the printout function. The program will then print the data entered.

After printing, all *malloc()* calls will be freed.

Call the file hw1.c. Use the *stdio* and *stdlib* header files.

OUTPUT

This is what your console output should look like if you programmed the assignment correctly:

```
Welcome to the Assignment 1! This program will add users to our database!

****** PROGRAM START *******

Please enter a number of users you would like to add: r
Please enter an integer! Try again: 2

****** New User 1 ******

        Enter user number: 1
        Enter Username: Syntax
        Enter First Name: Adam
        Enter Last Name: Spanier

****** New User 2 ******

        Enter user number: 2
        Enter Username: Deadlock
        Enter First Name: Bob
        Enter Last Name: Smith


****** BEGIN PRINTOUT ******


****** User Number 1 Summary ******

        User number: 1
        Username: Syntax
        User First: Adam
        User Last: Spanier

****** User Number 2 Summary ******

        User number: 2
        Username: Deadlock
        User First: Bob
        User Last: Smith


****** BEGIN FREE ******
```

SPECIFICATIONS

1. **DEFINE**
   a. *WELCOME* – The welcome message in the output
   b. *START* – Program Start message
   c. *PRINT* – Begin printout message
   d. *FREE* – Begin free message
   e. *SIZE* – 20
   f. Any other strings you want to add here, you can

2. **STRUCT**
   a. *Name*: user
   b. Struct Vars:
      1. int userNum
      2. char * userName
      3. char * userFirst
      4. char * userLast

3. **PROTOTYPES**

Name	Return	Parameters	Purpose
printWelcome	void	None	Print a welcome message
getNumUsers	int	None	Get number of users to add, validate
clearStdin	int	None	Clear standard in
addUsers	int	int, struct user[SIZE]	Add users to the user structure array
addString	char *	char *	Create a string, take user input, return
addInt	int	int *, char *	Take user input for an integer
printUsers	int	struct user[SIZE]	Print all users in the users structure array
freeMemory	void	struct user[SIZE]	Free all memory allocations

4. **FUNCTIONS**

   a. ***printWelcome*** – print the *WELCOME* and *START* macros.

   b. ***getNumUsers*** – Declare and *integer* called *num*. Declare a *char* called *term*. Print. “Please enter a number of users you would like to add: “. Use a perpetual *for loop* (for(;;){}). In the loop: **if(scanf("%d%c", &num, &term) != 2 || term != '\n')** then print “Please enter an integer! Try again: “ and call clearStdin(). Else, break. Return *num*. NOTE: Use the exact bold code above for the if statement condition.

c. ***ClearStdin*** – **Use this exact code**:

```
int clearStdin(){
  int c = 0;
  while (('\n' != (c=fgetc(stdin))) && (c != EOF)) {
    if (c == EOF) break;
  }
  return 0;
}
```

d. ***addInt*** – The input params are (*int * num, char * prompt*). In the function, print "Enter user number: " Be sure to remember the tab at the beginning. Use *scanf* to read in the *console user input* to the *num* variable. There is no need to use the & character on the *num* variable as it's already a pointer.

e. ***AddString*** – The input param is (*char * prompt*). In the function, use *malloc* to allocate *SIZE* times the *size of* a *char* for a new *char** called *str*. Null terminate the *str* variable. Print the incoming *prompt* with appropriate tabs. Use *scanf* to read *user input* into the *str* variable. Return *str*.

f. ***addUsers*** – The input params will be (*int loopNum, struct user users[SIZE]*). In the function, declare an *integer* called *i*. Create a *for loop* that starts at 0, counts up to, but not including *loopNum*, increments *i* by 1 each iteration. In the loop:

print "****** New User 1 ******" where the number 1 counts up each iteration. Call *addint* using: 1) the *address* of the *userNum* of the *current iteration* of the *user array* and 2) the string "Enter User Number: ". The first arg sounds complicated, but if you think through it, *it's not that bad*. To get the *current user* from *users*, you use: *users[i]*. To get the *userNum* from that *user*, you use *.userNum*. To get the *address* of that variable, you use the & character before *users[i]*.

Still in the loop, call *addString* with the prompt "Enter Username: ". Catch the *output* into *users[i].username*. Repeat with the prompt "Enter First Name: " and catch into *userFirst*. Repeat again with the prompt "Enter Last Name: " and catch into *userLast*.

Outside the loop, set the user number for the user at loopNum (users[loopNum].userNum) to 0 and return 0.

g. ***printUsers*** – The input param is (*struct user users[SIZE]*). In the function, declare an *int* called *num* and set it to -1. Declare another *int* called *count* and set it to 0. Print the *PRINT* macro. Create a *while loop* that loops while *num* is not 0 and *count* is less than *SIZE*. In the while loop:

Set *num* to *users[count].userNum*. Create an if statement: if *num* is not 0, print five things:

1. "****** User Number 1 Summary ****** where the 1 counts up each iteration (use *count*),
2. "User number: [userNum]"
3. "Username: [userName]"
4. "User First: [userFirst]"
5. "User Last: [userLast]"

In each of the print statements, the *usernum*, *username*, *first* and *last* names should belong the *current user* (reference *count*). Be sure your output looks EXACTLY like the output depicted above. This means you'll need to add newlines and tabs. Outside the if statement, use *count++* to increment *count*. Outside the while loop, *return 0*.

h. ***freeMemory*** – The input param is (*struct user users[SIZE]*). This function is much like the *printUsers* function. In the function, declare an *int* called *num* and set it to -1. Declare another *int* called *count* and set it to 0. Print the *FREE* macro. Create a *while loop* that loops *while num* is not 0 and *count* is less than *SIZE*. In the while loop:

   Set *num* to *users*[*count*].*userNum*. Create an if statement: if *num* is not 0: *free userName*, *userFirst*, and *userLast*. Outside the if statement, use c*ount++* to increment *count*.

i. ***main*** – Declare an *int* called *num*. Declare a *struct user* called *users* with a *size* of *SIZE (use [SIZE])*. Don't overcomplicate this. We're just creating an array of users. Set the *usernum* of the user at the *SIZE index (users[SIZE].userNum)* to 0. Call the *printWelcome* function. Call *getNumUsers* and catch the output in *num*. Call *addUsers* with *num* and *users* as arguments. Call *printUsers* with *users* as the argument. Call *freeMemory* with *users* as the argument. Return 0.

PROCESS

To get this to work, you'll need to pay *very careful attention* to how I've explained it. A good way to do this is to take the algorithm I've provided for each function and break it down into a list of "to-do's" Then, write the function by going down the list. After you write one function, test it by calling it from an empty main method.

Be sure you get the *addUsers* function working well *before you work on the printUsers* and *freeMemory* functions. To test *addUsers,* user internal print statements and try printing elements from *main* after you've returned the users array. You can do this with something like:

```
printf("%d\n", users[0].userNum)
printf("%s\n", users[1].userName)
```

Keep in mind the *size* of your returned *users* array. If it only has one record in it, you can't use the 1 index.

You are more than welcome to create MORE macros if you want to. I don't care if you create a macro for every string you create. Just make sure the output EXACTLY matches the output above.

GO SLOW, BE CAREFUL.

RUN YOUR CODE A LOT.

GIVE YOURSELF TONS OF TIME.

EMAIL ME A LOT.