**CIS*2500 Intermediate Programming**

# Assignment #2: contactsDatabase

## Concepts: Binary Files, Dynamic Allocation

## 1.0 The task:

Your task is to write a program that stores and retrieves structures in a file on disk. The file of structures must remain on the disk once your program ends. You should be able to store structures to the file and retrieve from the file after restarting the program.

The record that you will be writing to file has the following structure:

```
struct contact {
  long empIdPosn;
  long firstNamePosn;
  long lastNamePosn;
  long emailPosn;
  long next;
};
```

Here, empIdPosn, firstNamePosn, lastNamePosn, emailPosn, are the positions in the file of the Employee Id (integer), First Name (string), Last Name (string) and Email (string). These "locations" will be required for writing and reading the information to be stored.

When you write the structures on file, you will need to store the Employee id, First Name, Last Name, and Email separately from the contact structure. In the file, you will write the contact structure first followed by the Employee id, First Name, Last Name, and Email. The only required information is the Employee Id – all other information is optional. The position "next" stores the location in the file where the next contact record is stored.

Note that **you are required to use dynamic allocation to store the 3 strings, first name, last name and email, so that the returned string is on a heap.**

The file can contain any number of records (i.e., information on any number of employees). The name of the file will be contactListA2.db and if the file does not exist then you program must create it (Hint: Use mode "a+").

Prerequisites:

---

Use the given header file called givenA2.h – this file contains the proper function prototypes, constants, and structures required for this assignment, also detailed below. These constants, struct and function prototypes are defined in givenA2.h header file.

---

```
struct contact {
   long empIdPosn;
   long firstNamePosn;
   long lastNamePosn;
   long emailPosn;
   long next;
};
```

```
/* this function takes a filename as its only parameter and stores employee records in it*/
void createContacts (char * fileName);

/* this function takes a filename as its only parameter and prints all employee records stored in it in the
same sequence as it was created*/
void readContacts (char * fileName);

/* this function takes a filename and an int value for employee id, searches for that id, prints the employee
record and returns 1 if found, does not print anything and returns 0 if not found*/
int searchContacts (char * fileName, int keyId);
```

## 1.1 The Interface (Input and Output):

Your program will start with an input interface that creates contacts as shown below:

**Do you wish to enter a new contact (Yes or No)?:**

**Employee Id:**
**First Name:**
**Last Name:**
**Email:**

The program will move to the next stage when you answer **No** to the first question. The answer must be No exactly – observe the upper case N and lower case o.

The second stage interface will display a menu with 4 options (1) User enters a choice of 1: It will first read all contacts saved on the file and print them on the screen in the same sequence as they are stored in the file. A sample output is given next. (2) User enters a choice of 2: It will search the file for a given employee id. (3) User enters a choice of 3: It will allow the user to create more contacts. (4) User enters a choice of -1:  exits the program if the user enters -1.

**Note that (1) the program menu that calls on these functions will be programmed in contactsA2.c, and that the other functions will be saved in separate files.** (2) **You may want to open the file for creating contacts (stage 1 or option 3 of stage 2) in "a+" mode.**

The program will create the file contactListA2.db if it does not exist upon program start up.  If it does exist then it will be opened for reading and writing and the current file pointer should be positioned at the end of the file **(Hint: use mode "a+" when opening the file).**

**Sample Input / Output**

```
$ ./myContacts contactsListA2.db

Do you wish to enter a new contact (Yes or No)?: Yes

Employee Id: 10

First Name: Henry

Last Name: Wang

Email: hwang@uoguelph.ca


Do you wish to enter a new contact (Yes or No)?: Yes

Employee Id: 20

First Name: Hermaine

Last Name: Nag

Email: hnag@uchamplain.com


Do you wish to enter a new contact (Yes or No)?: Yes

Employee Id: 30

First Name: Gloria

Last Name: Mensah

Email: gmensah@utm.ca


Do you wish to enter a new contact (Yes or No)?: Yes

Employee Id: 12

First Name: Tiffany
```

```
Last Name: Zinga
Email: tzinga@uoguelph.ca

Do you wish to enter a new contact (Yes or No)?: Yes
Employee Id: 100
First Name: Mohit
Last Name: Kumar
Email: kmohit@utm.ca

Do you wish to enter a new contact (Yes or No)?: No

Enter a choice:
1 to print, 2 to search, 3 to create more, -1 to exit: 1

Printing Employee records saved on file:
****************************
Employee Id: 10
First Name: Henry
Last Name: Wang
Email: hwang@uoguelph.ca
****************************
****************************
Employee Id: 20
First Name: Hermaine
Last Name: Nag
Email: hnag@uchamplain.com
****************************
****************************
Employee Id: 30
First Name: Gloria
Last Name: Mensah
Email: gmensah@utm.ca
****************************
```

```
****************************
Employee Id: 12
First Name: Tiffany
Last Name: Zinga
Email: tzinga@uoguelph.ca
****************************
****************************
Employee Id: 100
First Name: Mohit
Last Name: Kumar
Email: kmohit@utm.ca
****************************

Enter a choice:
1 to print, 2 to search, 3 to create more, -1 to exit: 2

Search for Employee Id? 12

****************************
Employee Id: 12
First Name: Tiffany
Last Name: Zinga
Email: tzinga@uoguelph.ca
****************************

Enter a choice:
1 to print, 2 to search, 3 to create more, -1 to exit: -1

Exiting
$
```

**Note:** For choice 2, your program should prompt the user for the employee id of a contact they wish to search. The employee id must be a valid positive integer between 1 and 9999, your program must repeatedly prompt the user until a valid employee id is entered **(hint: validate**

**employee id in main).** Note that function searchContacts prints employee information if a match is found (as shown above) and returns 1 to the calling program (i.e. main), and 0 if it does not find a match. In such cases, main prints

*No match found.*

## 1.2 Details on storing information in the binary file

Based on the sample input/output scenario provided in section 1.1, the file contactsListA2.db will contain five records (of Employee ids 10, 20, 30, 12, 100). The data stored is as follows (**note that the following is only for demonstration – your program doesn't need to print it**).

Contact Record 1: stored at file position 0

- Employee Id posn = 40
- First name posn = 44
- Last name posn = 50
- Email posn = 55
- Record.next = 73

Contact Record 2: stored at file position 73

- Employee Id posn = 113
- First name posn = 117
- Last name posn = 126
- Email posn = 130
- Record.next = 150

Contact Record 3: stored at file position 150

- Employee Id posn = 190
- first name posn = 194
- last name posn = 201

- email posn = 208
- Record.next = 223

Contact Record 4: stored at file position 223

- Employee Id posn = 263
- first name posn = 267
- last name posn = 275
- email posn = 281
- Record.next = 300

Contact Record 5: stored at file position 300

- Employee Id posn = 340
- first name posn = 344
- last name posn = 350
- email posn = 356
- Record.next = 370

---

## 2.0 Compilation

Your program will be compiled using a makefile with the following flags:

```
-std=c99 -Wall
```

If you do not plan on implementing every function, stub it out (refer to your lectures to understand what a function stub is). Otherwise, your submission will not compile with our grader and you will get a 0.

---

### 2.0.1 Files required:

- givenA2.h — although you are given this file, you must submit this.
- contactsA2.c - This file must contain a main(). You must submit this.

- createContactsA2.c - You must submit this.
- readContactsA2.c — You must submit this.
- searchContactsA2.c - You must submit this.
- Makefile - You must submit this.

  The main target in your makefile must be called myContacts. Note that your makefile is required to have a target called clean, that allows to remove all object files and the executable file (myContacts). Read section 2.0.2 for more details on makefile.

---

## 2.0.2 Required folder structure for this assignment.

**/*\ Note that this is a new requirement (was not required for L1, L2 or A1).**

A2 folder tree structure (assume that the current folder is called A2)
A2 $

src include bin

Folder src must contain your source files (i.e., contactsA2.c, readContactsA2.c, searchContactsA2.c).
Folder include must contain your header file (i.e., givenA2.h)
Folder bin must contain your executable file (i.e. myContacts)
Note that the makefile should be located in the top-level directory (e.g. in folder A2). The commands in makefile must call your source, header and executable files via relative path. For example, the dependencies and commands to compile contactsA2.c using the above folder structure will be:

```
contactsA2.o: src/contactsA2.c include/lastnameFirstnameA2.h
	gcc -Wall -std=c99 -c src/contactsA2.c
```

Also note that all .o object files are created in the current folder (i.e. in folder A2), whereas the executable is created in folder bin. Remember this when you write the target for clean.

---

## 2.0.3 Running

Running such a program would be done as follows:

```
./bin/myContacts  <filename>
```

For example,

```
./bin/myContacts contactsListA2.db
```

---

## 3.0 Testing

In contactsA2.c file (that contains main and other function calls), you are still strongly encouraged to test throughout the entirety of the development lifecycle of this program.

---

## 4.0 Submission

You will submit your assignment files (createContactsA2.c, readContactsA2.c, searchContactsA2.c, contactsA2.c makefile, and givenA2.h) to A2 submission folder on GitLab. Read / Watch the tutorial on gitlab to recall the process of submission.

**Add a program header comment to each of your source files and the header file signifying your academic integrity. It must be of the following form:**

```
/*
  Student Name: Firstname Lastname
  Student ID: #######
  Due Date: Mon Day, Year
  Course: CIS*2500

  I have exclusive control over this submission via my password.
  By including this header comment, I certify that:
   1) I have read and understood the policy on academic integrity.
   2) I have completed Moodle's module on academic integrity.
   3) I have achieved at least 80% on the academic integrity quiz
  I assert that this work is my own. I have appropriate acknowledged
  any and all material that I have used, be it directly quoted or
  paraphrased. Furthermore, I certify that this assignment was written
  by me in its entirety.
*/
```

---

## 5.0 Marking

- Programs that don't compile receive an immediate zero (0).
- Programs that produce compilation warnings will receive a deduction of 1 mark per unique warning (i.e., 3 'unused variable' warnings will only deduct 1 mark).
- Loss of marks may also result from poor style (e.g., lack of comments, structure)
- Programs that use goto statements receive an immediate zero (0)
- Programs that use global variables statements receive an immediate zero (0)