Assignment 3

COMP 2401

Due: on November 25, 2018 before 23:55

Submission: Electronic submission on cuLearn.

1 Assignment Information

1.1 Objectives:

a. Using structures and unions.
b. Declaring variables as sub elements using bit fields
c. Formatting output using printf and sprint functions
d. Using string functions (e.g., strcpy(), strcmp())
e. Pass by reference (an address)
f. Coding small function
g. Using function pointers
h. Linked list operations

1.2 Assignment Grade

The grade of this assignment is out of 100 out of 110

Section 5 is a bonus section and is worth 40 points

You will be graded with respect to what you have accomplished. Namely, you earn points for working code and functionality.

1.3 Code Examples

1) Example code for data i/o formatting – the code provides examples of formatting output including left and right justification, spaces, floating point formats etc.
2) Sample code for linked list is provided

1.4 Coding Instructions:

1. Comments in Code – as provided in the slides given in class
2. No usage of global variables. All data must be passed or received via function parameters.

3. Write short and simple functions.

2 Submission (20 pts.)

- Submission must be in cuLearn by the due date.
- Submit a single tar file with all the c and h files. The file name is a3.tar
- Submit a Readme.txt file explaining
    - Purpose of software
    - Who the developer is and the development date
    - How the software is organized (partitioned into files)
    - Instruction on how to compile the program (give an example)
    - Any issues/limitations problem that the user must be aware of
    - Instructions explaining how to use the software

Grading of submission

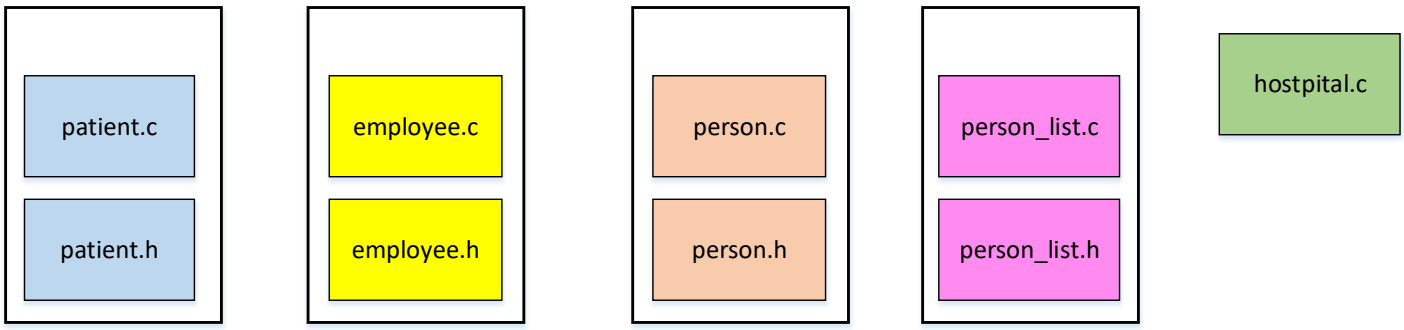
- 10 points for a readme file
    - 1 point for Purpose of software
    - 1 point for the developer name and development date
    - 2 points for description on how the software is organized (partitioned into files)
    - 2 points for instruction on how to compile the program (give an example)
    - 1 point for any issues/limitations problem that the user must be aware of
    - 2 points for instructions explaining how to use the software
- 2 points for proper submission of files
    - submitting a single tar file and a single readme file
    - using correct file names
- 8 points for using good program layout
    - Adding small functions as needed
    - Easy to read/review code
    - Code documented properly (e.g., purpose of functions, role of variable/parameters, code flow)
    - No “mega” functions
    - No global variables
- Submitting messy code may result in deduction of points.

3 Background

You are tasked to create the foundation of a system for manipulating records of patients and employees in a hospital. The final system would allow hospital administrators to query the data and to obtain some summary information. However, since this is the initial version the functionality is limited.

3.1 Program files layout

The system designer decided to split the functionality across multiple files. Each file would hold functions that operate on the structures.



1. File hospital.c – containing the main program (containing main(argc, argv)). The main function already contains a set of functions for testing the functionality. The main function also contains a testing function for populating an initial list of employees and patients.
2. Files patient.c and patient.h – file patient.c will contains all functions for handling patient records. In this initial version it will contain a single function for printing a patient record. File patient.h will contain the prototype of the patient functions.
3. Files employee.c and employee.h – file employee.c will contains all functions for handling employee records. In this initial version it will contain a single function for printing an employee record. File employee.h will contain the prototype of the employee functions.
4. File person.h person.c– file person.h contains the declaration of the three structures (patient, employee and person). It also contain some #define statements for the different constants that you may find useful in the code. File person.c will contain all functionality for processing a person record. In the initial version this file has no functionality. In future versions it will contain a function to print a patient record.
5. File person_list.h person_list.c– these files contain functionality for manipulating the list of hospital patients and employee. File person_list.h contains the declaration and prototypes of the functions. File person_list.c contains the implementation of the functions. Note that several helper functions are declared but are not required to be coded. The current version of the list manipulation functions are strongly

coupled with the patient and employee records. Future versions are aimed at decoupling the functionality using functions pointers.

Note that the files contain additional prototype for short and simple functions. You may find it useful to code them but you do have to as part of this assignment.

6. Files populateRecord.c and populateRecord.h, initial_list.o and inial_list.h– these files are part of a test suite and are not part of the program that you must code. The functionality is already coded. They populate the initial linked list. These files should not be modified.
7. Testing code files – four additional code files are provided for testing purpose. You should not modify these files. The files populateRecord.c and populateRecord.h are used to populate a person record (patient or an employee). The file initial_list.h is the header file of the function used to create the initial linked list. The file initial_list.o is the object file containing the function createInitialList().
8. Compiling the final executable requires a compilation and linking of multiple files. You may want to consider creating a Makefile to manage the code.

   gcc hospital.c person.c person_list.c employee.c patient.c populateRecord.c initial_list.o

3.2 Structures

The data of a patient or an employee is initialized using the function populateRecord(). The data is stored in a list and the size of the array is given using the define NUM_RECORDS. The list is created using the function createInitialList()

The patients and employees data:

a. Shared data:

   First Name – first name

   Family Name – family name,

   Telephone – telephone number

b. Employee only data:

   Salary – annual salary

   Years of service – number of years at the hospital

   Position – the position at the hospital

   Department – the department that the doctors is working at.

c. Patient only data:

   Department - the department the patient is in

   Daily cost – cost of having the patient in the hospital for one day

   Number of days – number of days that the patient is hospitalized

   Severity – the seriousness of the patient illness.

The following information defines the records fields:

Common fields

- First Name – 14 characters long (not including the '\0' character)

- Family Name – 14 characters long (not including the '\0' character)

Patient fields use bit fields to minimize the size

- Department – an integer in the range of 1-6
- Daily cost – an integer number in the range of 1-50
- Number of days in hospital – an integer number in the range of 0-30
- Severity – an integer number in the range of 0-3

Employee fields use bit fields to minimize the size

- Salary – a real non negative number
- Years of service – an integer in the range of 0-60
- Position – an integer in the range of 0-3
- Department - range 1-6 (integer)

4 Tasks (90 pts.)

In this assignment you will write a short program to manipulate person data of a hospital: patients and employees).

Suggestions

- As you code your small helping functions write small test functions to ensure that the code is correct. This will allow you to focus on the logic of your program without worrying about the simple functions. In particular make sure that you have a few functions that check the validity of the input.
- Create function for each menu option. The function should accept as input the array of person and the number of elements in the array.

Notes

The program creates a linked list of size NUM_RECORDS persons (currently set to 20). It does it by invoking the function createInitialList(). This is one of the test suite functions. You will need to use populateRecord.c (file populateRecord.h contains that function prototype) and initial_list.o (initial_list.h contains the function prototypes) when you compile and link these files with your program.

4.1 Modifying patient, employee, and person structures (15 pts.)

Initial design of the data structures determined that three data structures are sufficient: PatientRec - patient

specific data, EmployeeRec - employee specific data and, PersonRec – person data that is common to both.

```
// structure contains patient information
typedef struct patient {
    int department;                 // department in hospital
    int dailyCost;                  // cost of hospitalization per day
    short numDaysInHospital;        // number of days in hospital
    char severity;                  // severity of illness
} PatientRec;

// structure contains employee information
typedef struct employee {
    int position;                   // position of employee in hospital;
    int yearsOfService;             // years of service
    int department;                 // department in hospital
    float salary;                   // annual salary
} EmployeeRec;

// structure contains person information
typedef struct person {
    char firstName[NAME_SIZE];
    char familyName[NAME_SIZE];
    char employeeOrPatient;
    union {
            EmployeeRec emp;
            PatientRec patient;
    };
} PersonRec;
```

The program can determine whether the PersonRec is a patient record or an employee record using the discriminant field employeeOrPatient. If the employeeOrPatient == EMPLOYYE_TYPE the PersonRec contains an employee data. If the employeeOrPatient == PATIENT_TYPE the PersonRec contains patient data.

The type of many of the fields does not correspond to the value that a field must hold. Examples are:

- Family Name – in PersonRec the family name can be at most 14 characters long. Storing the '\0'

character with the family name means that only 15 characters are required for the field familyName. However, the designer allocated 64 characters to the family name (the designer set NAME_SIZE to 64). Here you can reduce the size by setting the NAME_SIZE to 15

- In PatientRec the severity field can have values 0-3 which require only 2 bits. However, the initial design set the field severity to an int which is 32 bits. Here you can reduce the size to 2 bits for this field.

In this task you need to modify each structure so that:

- Each field can store all the values in the required range
- The structure requires as little memory space as possible.

In order to achieve it you can:

- Change the type of each of the fields in the structure
- Change the order of each of the fields in the structure
- Use bit fields to create subtypes of the integer family of data types - char, short, int or long (either signed or unsigned).

  You cannot change the fields' names, or the structures names.

Grading (this includes some bonus points)

- 5 points for minimizing the structure sizes
- 10 pointes – for using proper packing order and bit fields.
  - 3 points for packing the structure in the correct order
  - 7 points for using bit fields correctly (including data types)

4.2 Print Employee Record (10pts)

Code a function to print a single employee as defined in the format below.

Hint: use sprintf() to print the first name and the last name into a temporary string. Then use the temporary string to print the name as defined in the formatting instructions below.

The computation of the salary-to-date is yearsOfService x salary.

The "xxx" in the print format represent the number of characters that the printed value must occupy. For example depts.: is followed by xx. Here the dept: number has to occupy 2 spaces. If the value does not have 2 digits (e.g., 14) but rather one digit (e.g., 1) than a space must be used the 1such as dept: 1

Print format

First Name Family Name (33 char) dept:xx salary:xxxxxx.xx position:xx years of service:xxxx salary to-date:xxxxxxxx.xx

For example:

Don Johnson dept: 5 salary: 35510.00 position: 2 years of service: 17 salary to-date: 603670.00

Printout example

Hospital Employees

Don Johnson dept: 5 salary: 35510.00 position: 2 years of service: 17 salary to-date: 603670.00

John Ouster dept: 2 salary: 7644.70 position: 3 years of service: 47 salary to-date: 359300.91

Function prototype
void printEmployee(PersonRec *person);

Grading:

- 10 points for printing as required

4.3 Print Patient Record (10pts)

Code a function to print a single patient as defined in the format below. Hints use sprintf to print the first name and the last name to a temporary string. Then use the temporary string to print the name as defined in the formatting instructions below.

The computation of total cost is daysInHospital* dailyCost

First Name Family Name (33) characters dept:xx days in hospital:xxx severity:xx daily cost:xxx total cost:xxxxx

John Johnson dept: 3 days in hospital: 21 severity: 0 daily cost: 32 total cost: 672

Output example

Patient List

John Johnson dept: 3 days in hospital: 21 severity: 0 daily cost: 32 total cost: 672

David Carp dept: 5 days in hospital: 26 severity: 1 daily cost: 37 total cost: 962

Function prototype

void printPatient(PersonRec *person);

Grading:

- 10 points for printing as required

4.4 Print All Persons using a Function Pointer(15pts)

a) Coding the function printListFun() - Code a function to print all the records in the input linked list using a function pointer. The function needs to traverse the linked and print each record in order. The data in the node should be passed to the function printFun() (the input function pointer parameter).

Function prototype

void printListFun(PersonList *head, void (*printFun)(PersonRec *));

b) Code that prints a person - In order to use the function printListFun() one must provide the address of a print function as one of the parameters. Therefore, you need to create a function in the file person.c which can accept a parameter of type PersonRec * and then depending on the type of person invokes the corresponding function. Namely, if the person is an employee record then print the record using the function printEmployee() and if the record is a patient record then print it using the function printPatient().

Code the function in the file person.c

Add the function prototype to the file person.h

Function prototype

Function prototype

void printPerson(PersonRec *person);

c) Code a function to print only patients – Here you have to code a function that given a person record prints only patients. This can be easily done by modifying the pseudo code of the function printPerson() to print only patients. Namely, the function will accept a parameter of type PersonRec * and then if the record is a patient record then print it using the function printPatient(). The function will ignore the employee records.

Code the function in the file person.c

Add the function prototype to the file person.h

Function prototype

Function prototype

void printOnlyPatientsFun(PersonList *person);

Grading:

- 5 points for coding the printListFun function
- 5 points for coding the printPerson() function
- 5 points for printing the printOnlyPatientsFun()

4.5 Print All Persons by name (10pts)

Code a function that prints all the persons in the input linked whose family name matches the input family name. Here you need to compare each person name in the record with the input name. If the name matches that print it using he function printPerson() that you already coded.

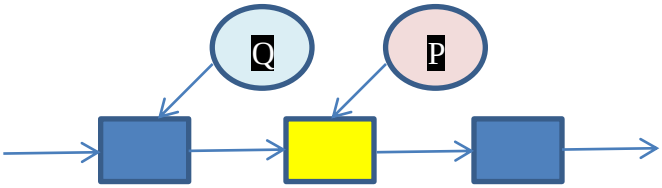
Function prototype

void printAllByName(PersonList *head, char* familyName);

4.6 Delete by Name (10pts)

Code a function to delete a person record from the linked list. The function needs to delete the first record in the linked list where the person's family name matches the input family name parameter.

Note, here the function first needs to find the first record in the list with a matching name. However, in order to be able to delete the node it needs to "remember" the node before the node with the matching name. In the example below assume that the yellow box represents the node with the matching name and that *P* points to it. In order to delete the node pointed to by *P* the function needs to "remember" the node pointed to by *Q*.



Here the function needs to search for a node with a matching name. There are three cases possible outcomes:

- Person not found – no person with a matching name exists
- Person with a matching name is the first in the list. In this case the function should delete the first node
- Person with a matching name is in the list. In this case the function needs to do deleteAfter operation on node Q.

The function needs to return the person stored in the node using the parameter data. In case the node is not found the function should return

Function prototype

int deleteNodeByFamilyName(PersonList **head, char *familyName, PersonRec **data);

Grading:

- 10 points for printing as required

4.7 Delete List (10pts)

Code a function that deletes all the nodes in the given list. The function **should not** free the memory of the person records stored in the nodes. It will be up to the calling program to do so.

Function prototype

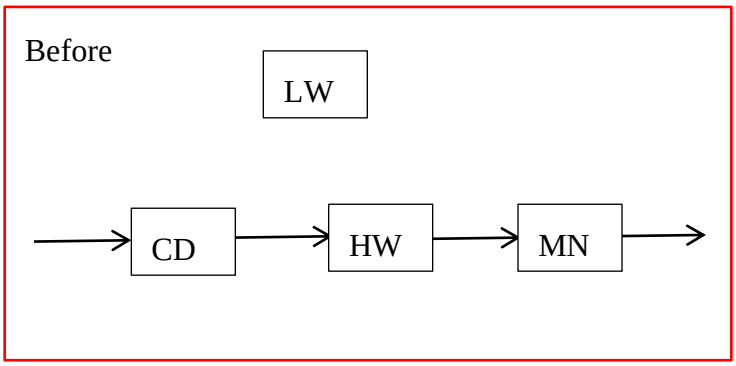
void deleteList (PersonList **head);

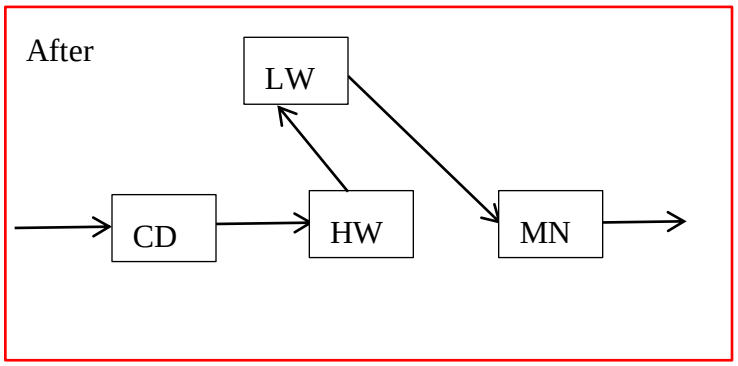
Grading:

- 10 points for printing as required

4.8 Insert By Name (10pts)

This function inserts a new person record into the linked list. Here the function needs to insert the new person in lexicographic order using the family name. For example if the linked list contains the persons CD, HW and MN and the new person is LW then the function needs to insert the new person between the person HW and MN (see before and after below).





Function prototype

int insertByName(PersonList **head, PersonRec *data);

Grading:

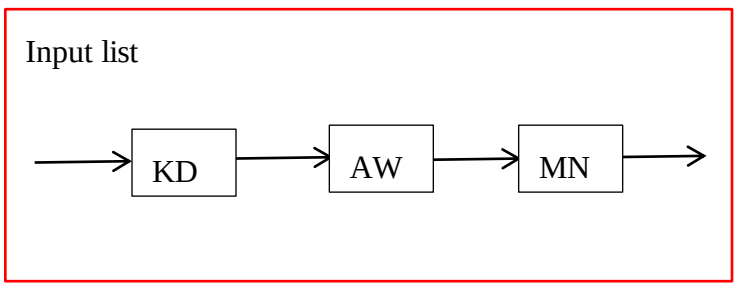
- 10 points for printing as required

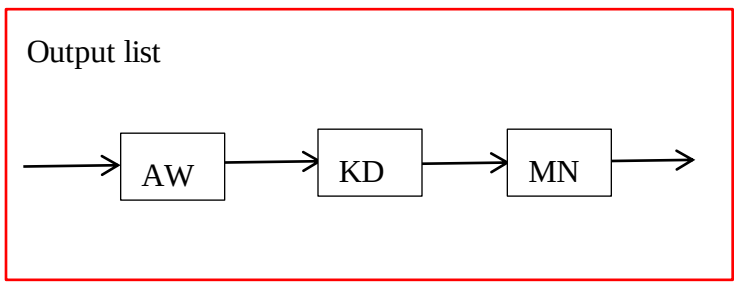
5 Bonus (40 pts.)

This section is a bonus section. You are not required to do it. It contains somewhat more challenging tasks.

5.1 Copy Sorted (20 points)

Here you are required to copy and input linked list into a new linked list. However, the output linked list should appear in sorted order by family name. The lexicographic order is by family name from smaller to larger. For example, if the input list is KD, AW and MN the output list should be AW, KD and MN





void copySorted(PersonList *inHead, PersonList **outHead);

Note:

The function **should not** allocated memory to the PersonRec stored in the data field of each node. Rather nodes in the two lists will share the PersonRec that its address is stored in the data field.

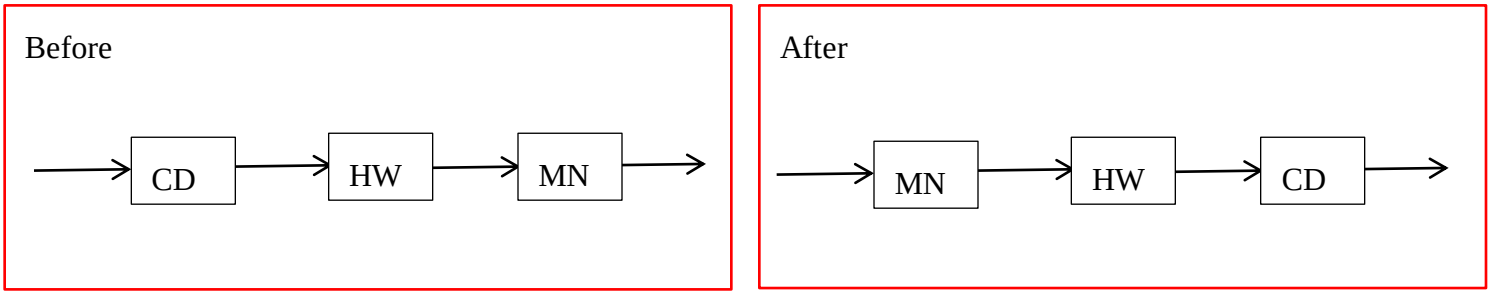
Code the function in the file person_list.c

Add a prototype in the file person_list.h

Clearly state in the readme file that you coded this function

5.2 Reverse List (20 points)

Here you are required to write a recursive function that reverses the order of the linked list. Namely, the first node in the list becomes the last node in the list; the second node in the list becomes the second last node in the list and so on. For example if the linked list is CD, HW, and MN after calling the function the linked list is MN, HW, and CD



In doing so you

1. are not allowed allocate new nodes or any new memory (no memory allocation)
2. are allowed to use local variables
3. are not allowed to write the linked list to or from a file

Code the function in the file person_list.c

Add a prototype in the file person_list.h

Clearly state in the readme file that you coded this function

Function prototype

void reverseList(PersonList **head);