**CPE 111 - Programming with Data Structures**
**International Sections - January 2021**
**Laboratory Exercise 3**

**Objective**

This lab is intended to give you practice coding a simple linked list. You will modify the program **couples.c** you created in Lab 1 to be able to handle an unknown number of people in the file, by using a linked list where one person structure points to the next one.

**Instructions**

Copy your program **couples.c** from Lab 1 to a new program called **linkCouples.c**. If you did not succeed in getting your program to work correctly, you can use the version of the program from the LabSolutions directory as a starting point, but I'd rather you used your own.

Like the old program, the new program will read a file with the names and genders of people, and create structures representing these people. However, instead of storing the people in a dynamically allocated array, it will create a linked list. Then the program will ask the user for pairs of people who are in a couple, and will store that information using pointers. Finally, the program will print out the names of all people in couples.

To run this program, you will type

```
./linkCouples inputfile.txt
```

Thus this program can be run with many different input files. The input files have information like the following sample **newpeople.txt** (which you can download from the website).

```
John M
Max M
Steven M
Bart M
Zeke M
Michael M
Louise F
Cynthia F
Jennifer F
Martha F
Iris F
Penelope F
Heather F
Ren F
Jeremy M
Helen F
Robert M
Supat M
Serena F
Anchalee F
Andrew M
Carol F
Charley M
Thomas M
Chaiwat M
```

Notice that unlike Lab 1, there is ***no count in the first line***, so you cannot know how many people are in the file. So instead of allocating an array, you will change the **PERSON_T** structure so that it can link to the next person in the list.

**What will change?**

1. You need to add a "next item" pointer to the structure to hold one person. For example:

```
typedef struct _person
  {
  char name[32];               /* person's name */
  char gender[2];              /* person's gender */
  struct _person * pPartner;   /* pointer to another PERSON_T who is this person's partner.*/
  struct _person * pNext;      /* pointer to the next person in the list */
  } PERSON_T;
```

2. You need to declare two **PERSON_T*** variables to hold the head and the tail of the list.

3. You need to delete the code that reads the count from the file (since there is no count).

4. You need to take delete the code that allocates the array of **PERSON_T*** records, since we don't know how many to allocate.

5. You need to change the file reading loop to have the following logic:

```
while fgets of next line does not return NULL
      parse the line to get the name and gender (using sscanf)
      allocate a new PERSON_T using calloc
      if the calloc returns NULL
         print an error message and exit
      else
         copy name and gender to the new person record
         add the new person record to the end of the linked list
      endif
endwhile
```

6. You need to change the code that loops through and prints the information about all the people. Instead of iterating through an array, this code needs to iterate through the linked list. The logic is as follows.

```
set thisPerson = head;
while thisPerson is not NULL
      print information for thisPerson
      thisPerson = thisPerson->pNext
endwhile
```

7. You need to change the code that finds a person based on his or her name. Instead of iterating through an array, this code needs to iterate through the linked list, checking the name on each person until it finds the person it is looking for, or else gets to the end of the list. *I strongly recommend that you put this into a function.* Then you can use the function for looking up both people in a couple.

```
set foundPerson= NULL
set thisPerson = head;
while thisPerson is not NULL
      if thisPerson->name equals the search name -- use the strcmp function
         foundPerson = thisPerson
         break;
      endif
      thisPerson = thisPerson->pNext
endwhile
return foundPerson             -- if NULL, that means the person does not exist
```

8. The code that prints out the couples, at the end, also needs to iterate through the list.

9. The code to free the linked list also needs to iterate through the list. You need two **PERSON_T*** pointers to manage this.

```
set delPerson= NULL
set thisPerson = head;
while thisPerson is not NULL
     delPerson = thisPerson
     thisPerson = thisPerson->pNext
     free delPerson
endwhile
head = tail = NULL
```

The program should behave exactly the same as the original **couples.c**.

Upload **linkCouples.c**.

**Extra Challenge**

If you finish Lab 3 and would like to do some additional work that is a bit more difficult, try this!

- Before you enter the loop where you create couples, but after printing out all the people, ask the user whether he or she wants to delete any people from the list
- If the user says yes, enter a loop. In the loop ask for the name of the person to remove.

- Try to find the person in the list. If the person exists, remove him or her from the list. (You should create a function for this.)
- Ask if the user wants to delete more people. If the answer is yes, repeat the steps above.

A sample run might look like this:

```
Do you want to delete people? Y
Enter name to delete: Zeke
-- Zeke has been deleted. Continue? Y
Enter name to delete: Carol
-- Carol has been deleted. Continue? Y
Enter name to delete: Marissa
-- Marissa not found in the list. Continue? Y
Enter name to delete: Serena
-- Serena has been deleted. Continue? N
```

Then continue to make couples.

When you remove a person from the list be sure that you:

1. Adjust the "next" pointers of the items before and after the deleted person (if any)
2. Adjust the head pointer if you delete the first person in the list
3. Adjust the tail pointer if you delete the last person in the list
4. Free the memory for the person you remove.