# COIS 3380: Lab 4

## String comparison using recursion

Use your terminal to connect and login to Loki as you did in labs previously. Move to your 3380 directory using the `cd` command. Once you are inside your 3380 directory, create a new directory called "lab4" using the `mkdir` command. Navigate to the newly created "lab4" directory.

Using `mystr.c` on blackboard, we are going to complete the recursive function `myStrCmp()` to compare 2 strings. The input to this function will be the memory locations (pointers) of the 2 strings (`s1` and `s2`). You need to complete the code for the function such that it compares if the 2 strings are equal or not. If they are equal, it will return `0`. If not, then the function will return `-1` if the first string (`s1`) appears before the second string (`s2`) in lexicographical order, otherwise `1` if the second string appears before the first string in lexicographical order.

Memory location of the strings can be used as inputs for the function using the star (`*`) operator. The function header can be written somewhat this way:

```
int myStrCmp(char *s1, char *s2);
```

Where, `*s1` and `*s2` are the memory locations of string `s1` and `s2`.

While creating this function we will also assume that `s1` and/or `s2` will not be `NULL`.

For example: if `s1 = "Monday"` and `s2 = "Monster"` then a call to `myStrCmp(s1, s2)` would return `-1` because `Monday` should appear before `Monster` in a lexicographic order.

To compare the current characters of a string, we will use the * operator once again. For instance, `*s1 == *s2,` will compare the first characters of string `s1` and `s2` (similar to s1[0] and s2[0]). To move to the next address that is stored by the pointers, we can use the '++' operator.

```
s1++;
```

```
s2++;
```

will move the pointers' location to point to the location of the next character.

To check if a string is empty, you can check if the current character pointed by *s1 or *s2 is a '`\0`'. This will allow you to know if all the characters in a string have been checked or not.

The algorithm to compare the string will be something like this:

1. If both strings end with the '\0' then return 0.
2. If not, compare the strings and if two characters of the strings are not equal, then return the results of their comparison
3. Increment the pointers and call the function once again.

*Note: Be sure to update the pointers location before the recursive function call.*

## Initilization and searching an array

In this part of the lab, we will focus on passing structures in C as pointers and accessing its fields. In `find_struct.c` on blackboard, there is an outline of a program that can be used to find an employee from a list of employees.

In the program, we first created a `struct` called `emp`. A struct in C is a user-defined data type that allows us to group different types of variables together under a single name. In our case, the `emp` struct will be used to store the first name, last name, and employee Id of a particular employee. In the `main()` function, 2 arrays are created with their allocated size of `MAX_EMPLOYEES`. `empArr` contains the emp struct, while `empPtrArr` contains pointers to the employee structure.

Employee information is added to `empArr` using the `populate()` function. The empArr is sent as a reference to the `populate()` function where an employee first name, last name, and employee Id are added. To store the first name and last name we use the `snprintf()`. This function allows us to store a series of characters and its values into a string. They syntax for `snprintf()` is as follows.

```
int snprintf(char *str, size_t size, const char *format, ...);
```

where,

- `str`: A pointer to the string where the formatted string will be stored.
- `size`: The maximum number of characters to be written to the string, including the null terminator (‘`\0`’).
- `format`: A format string that specifies how to format the subsequent arguments.
- `...`: The values to be formatted and inserted into the resulting string.

Additionally, we use the arrow `(->)` operator to access members of a struct through a pointer. This differs from the dot `(.)` operator, which is used to access members of a struct directly.

Your task is to complete the 2-remaining function `compareEmployee()` and `findEmployee()`. `compareEmployee()` accepts a pointer to the `emp struct`, which contains the employee details , and a string `keyLastName`. You need to complete this function by comparing the last name of the employee (from the struct) with the `keyLastName`. The should return 0 if the names match, or a non-zero value if they do not.

Moreover, you also need to complete the `findEmployee()` function. It accepts an array with the pointers to the `emp struct`, that contains the list of the employees, an integer called `arraySize` that contains the number of elements in the array, and finally a string called `LastName` which represents the last name that will be searched for. The function should iterate through the array comparing each employee’s last name with the provided `LastName`. If an

employee with the given last name is found, then it should return a pointer to that `emp struct`. If no employee is found, then the function should return `NULL`.

## Demonstration

Please show the lab demonstrators the implementation and output of `mystr.c` and `find_struct.c` to receive your grade.