Data Structure Lab Program - 2016

Lab exercise – 3

Note: All programs must be written in C following coding rules as mentioned on the course website.

1. Read a square matrix of size n.
   a. Print all unique values along each row.
   b. Print all unique values along each column.
   c. Print all unique values in the matrix.

2. Create an array of employee names.
   a. Arrange them in lexicographically sorted order.
   b. Print all unique names.

3. Perform following operations on string:
   a. Print the length of a given string.
   b. Concatenate two strings.
   c. Check whether the string is palindrome or not.
   d. Reverse the given string.

4. Implement abstract data type *stack* using array. It should support the following operations:
   a. push: Inserts a given element into stack.
   b. pop: Removes and returns the last element inserted into the stack.
   c. top: Last element inserted into the stack.
   d. size: Number of elements in the stack.
   e. is_empty: True if the stack if size is zero and false otherwise.
   f. is_full: True if the stack is full and false otherwise.

5. Implement stack as a `struct` given below. Solve all the problems described in Q4a-f.
```
struct stack {
        size_t size;
        int data[100];
};
```

6. Implement stack as a `struct` given in Q5. Write a function to duplicate stacks using the prototype given below. After copying the content of the two stacks must be the same.
```
struct copy_stack(struct stack);
.
.
struct stack s1;      // creation of stacks s1 and s2
struct stack s2;
.
.
s2 = copy_stack(s1);  // usage of the function
```

7. Implement stack as a `struct` as in Q5 with an additional member `capacity` of type `size_t`. Instead of `data` as an array of 100 elements, use a pointer for a dynamically created array (`int *data`) with a specified size (`capacity`) provided with an additional operation `create`.