Section 6 **Autumn-2024**

PDS Lab **Lab-8** **15.10.2024**

Instructions:

- This lab is based on the topics: Recursion, Searching, Sorting.
- You should save each program with the file name as specified against each problem as <Lab#>-<Assignment#>-<Roll#>.c. For example, **08-01-24NA10006.c** to save Program to 1st assignment in Lab 6 with Roll Number 24NA10006
- You should upload each program to the Moodle system. Also, copy + paste your programs to the text window on the test page.
- A few test cases against each problem are given for your reference, including but not limited to.
- There are three problems and the maximum time allowed is 150 minutes.
- **Do not use pointers and structures in this lab.**

Problem 1: Recursion

Write a C program that:

a. Reads a natural number **n** from keyboard. ($n \in [0, 10^5]$) [2]
b. Calls a **recursive** function **digits()** to find the number of digits in **n**. [10]
c. Calls a **recursive** function **IsPalindrome()** to check if **n** is a palindrome. [15]
d. Prints whether the input is a palindrome or not. [3]

Definition: A number is a palindrome if it is the same when read in reverse order, i.e, from right to left. Eg. 121.
(NOTE: Non-recursive function will not be awarded any marks.)

Test cases:

#	INPUT	OUTPUT
1	474	Palindrome
2	513	Not palindrome
3	0	Palindrome
4	2124	Not Palindrome

[2 + 10 + 15 + 3 = 30]

```c
//  Check if a number is a palindrome or not using recursion
//  Created by Anshita Gupta on 14/10/24. (anshitagupta.ag@kgpian.iitkgp.ac.in)
#include <stdio.h>
#include <math.h>

// Recursive function to find the number of digits in n
int digits(int n){
    if (n == 0)
        return 0;
    return 1 + digits(n / 10);
}

// Recursive function to check if n is a palindrome
int IsPalindrome(int n, int length){

    // Base case: When the number becomes 0
    if (n == 0)
        return 1;

    int firstDigit = n / (int)pow(10, length - 1);
    int lastDigit = n % 10;

    // Condition if not a palindrome
    if (firstDigit != lastDigit)
        return 0;

    // Checking the remaining number
    n = (n % (int)pow(10, length - 1)) / 10;
    return IsPalindrome(n, length - 2);
}

int main(){
    int n;
    scanf("%d", &n);
    // Calling recursive function to find the number of digits in n
    int length = digits(n);
    // Calling recursive function to check if the number is a palindrome
    if (IsPalindrome(n, length))
        printf("Palindrome\n");
    else
        printf("Not Palindrome\n");
    return 0;
}
```

Problem 2: Searching & Sorting

Write a program that does the following:

- **a.** Read the size of input array as an integer `n`. ($n \in [0, 100]$) [2]
- **b.** Read the array of size `n` in $A[a_1, a_2, \ldots, a_n]$. ($a_i \in [-10^5,\ 10^5]$) [3]
- **c.** **Sort** the array `A[]` using any sorting technique. ($(a_i \leq a_{i+1})\ \forall\ i \in [0, n-2]$) [10]
- **d.** Read an integer `m` from the user. ($m \in [-10^5,\ 10^5]$) [2]
- **e.** **Insert** `m` in the sorted array while maintaining order. [10]
- **f.** Print the sorted array. [3]

(NOTE: The value `m` must be **inserted** into the array `A[]` i.e. `A[]` must be modified.
No marks will be awarded if `m` is just printed in the correct position, i.e. without inserting)

Test cases:

#	INPUT	OUTPUT
1	n = 5 A[n] = 15 5 10 25 20 m = 17	Output Array: [5, 10, 15, 17, 20, 25]
2	n = 5 A[n] = 15 5 10 25 20 m = 30	Output Array: [5, 10, 15, 20, 25, 30]
3	n = 5 A[n] = 10 -2 4 6 8 m = 5	Output Array: [-2, 4, 5, 6, 8, 10]
4	n = 5 A[n] = 10 -2 4 6 8 m = 0	Output Array: [-2, 0, 4, 6, 8, 10]

[2 + 3 + 10 + 2 + 10 + 3 = 30]

```c
// Sort and insert
// Code creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
#include <stdio.h>
#define MAX 100

// function to sort the array using bubble sort
void sort(int arr[], int n)
{
    int i, j, temp;
    for (i = 0; i < n; i++){
        for (j = i + 1; j < n; j++){
            if (arr[i] > arr[j]){
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
void print_array(int arr[], int n){
    printf("\n[");
    for (int i = 0; i < n-1; i++)
        printf("%d, ", arr[i]);
    printf("%d]\n", arr[n-1]);
}
int main(){
    int n, m;
    scanf("%d", &n);
    int arr[MAX]; // array of size 100
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    // sort the array
    sort(arr, n);
    scanf("%d", &m);
    int i = 0;
    // find the position where m should be inserted using linear search
    while (i < n && arr[i] < m)
        i++;
    // shift the elements to the right
    for (int j = n; j > i; j--)
        arr[j] = arr[j - 1];
    // insert m
    arr[i] = m;
    n++; // increase the size of the array
    // print the array
    print_array(arr, n);
    return 0;
}
```

Problem 3: Sorting

Write a program that does the following:

a. Read the size of input array as an integer `n`. ($n \in [2, 100]$) [1]

b. Read the array of size `n` in $A[a_1, a_2, \ldots, a_n]$. ($a_i \in [0,\ 10^5]$) [2]

c. Modify the array `A[]` such that the largest possible number that can be formed using the elements of $A[a_1, a_2, \ldots, a_n]$ appened together to each other. [35]

For example:

A[5] = [5, 10, 9, 97, 4]
Largest = 9975410
New A[5] = [9, 97, 5, 4, 10]

d. Print the modified array and the largest number. [2]

(NOTE: The array `A[]` must be modified.
No marks will be awarded if the largest number is just printed without modifying the array)

Test cases:

#	INPUT	OUTPUT
1	4 1 1 1 1	A[4] = [1, 1, 1, 1] Largest = 1111
2	4 1 1 2 3	A[4] = [3, 2, 1, 1] Largest = 3211
3	6 10 685 75001 7 21 12	A[6] = [7, 75001, 685, 21, 12, 10] Largest = 775001685211210
4	10 685 97005 9 21 12	A[6] = [9, 97005, 685, 21, 12, 10] Largest = 997005685211210

$[1 + 2 + 35 + 2 = 40]$

```c
// Largest Number Formed by Concatenating Numbers
// Code Creator: Saurav (sauravgahlawat@kgpian.iitkgp.ac.in)
#include <stdio.h>

// Swap function to swap two integers
void swap(int *a, int *b)
{
    int temp = *b;
    *b = *a;
    *a = temp;
}

// Compare two numbers based on which concatenation is larger
int check_swap_needed_by_comparing_digits(int a, int b)
{
    long ab = 0, ba = 0, multiplier_a = 1, multiplier_b = 1;

    // Calculate the ab = a concatenated with b
    int temp_a = a, temp_b = b;
    while (temp_b > 0)
    {
        multiplier_a *= 10;
        temp_b /= 10;
    }
    ab = a * multiplier_a + b;

    // Calculate the ba = b concatenated with a
    temp_a = a;
    while (temp_a > 0)
    {
        multiplier_b *= 10;
        temp_a /= 10;
    }
    ba = b * multiplier_b + a;

    // Return which concatenation is larger
    if (ab > ba)
        return 1;
    return 0;
}

int main()
{
    int n;
    scanf("%d", &n);
    int arr[n];

    // Input the array of numbers
```

```
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    // Sort the array manually using the compare_digits function
    for (int i = 0; i < n; i++)
        for (int j = 1; j < n - i; j++)
            if (check_swap_needed_by_comparing_digits(arr[j], arr[j - 1]))
                swap(&arr[j], &arr[j - 1]);

    // Print the array
    printf("A[%d] = [", n);
    for (int i = 0; i < n - 1; i++)
        printf("%d, ", arr[i]);
    if (n > 0)
        printf("%d]\n", arr[n - 1]);

    // Print the largest number formed
    printf("Largest = ");
    for (int i = 0; i < n; i++)
        printf("%d", arr[i]);
    printf("\n");

    return 0;
}
```