Section 6 Autumn-2024

PDS Lab **Lab-5** **03.09.2024**

Instructions:

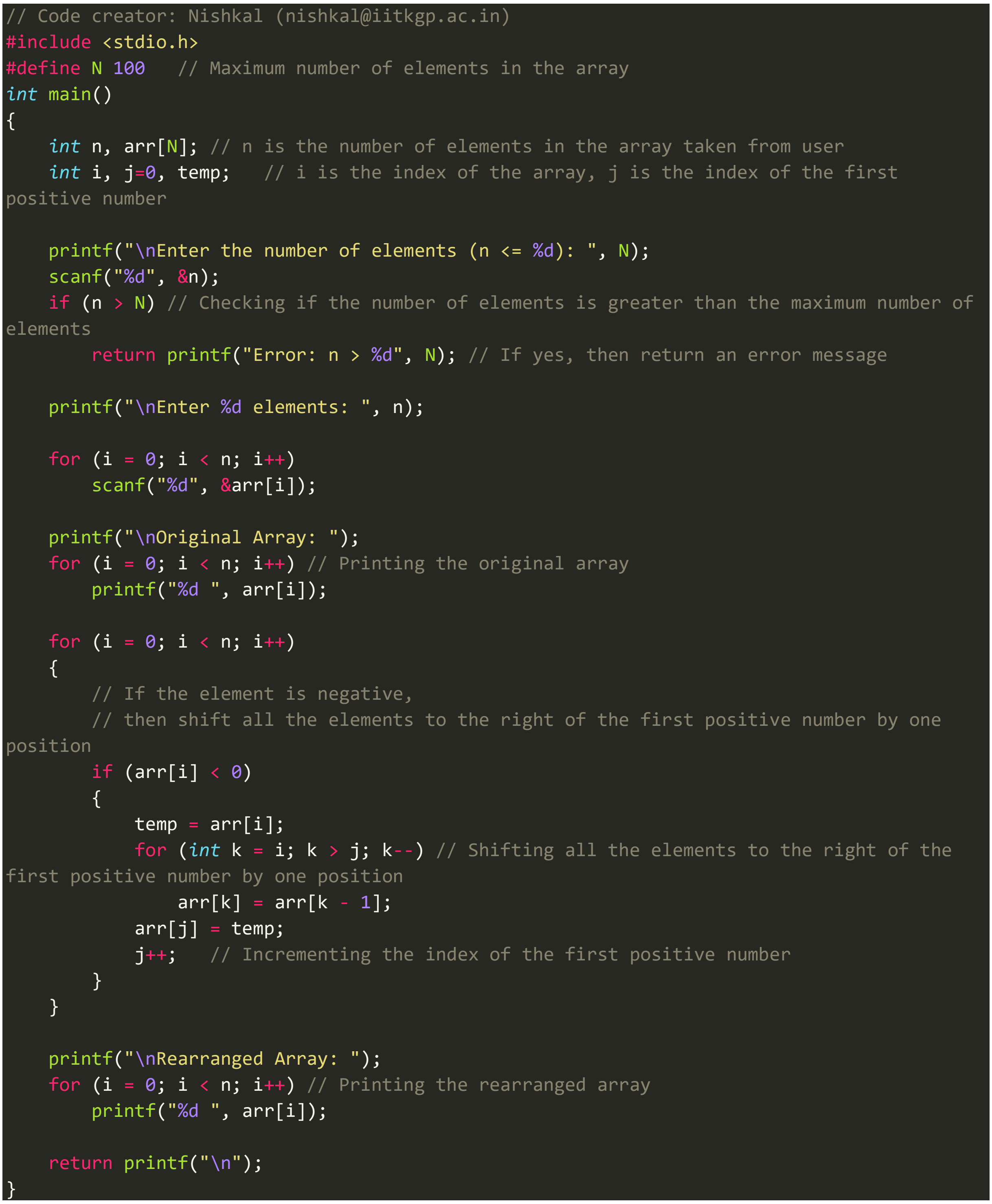
- This lab is based on the topics: 1D Arrays.
- You should save each program with the file name as specified against each problem as <Lab#>-<Assignment#>-<Roll#>.c. For example, **05-01-24NA10006.c** to save Program to 1st assignment in Lab 5 with Roll Number 24NA10006
- You should upload each program to the Moodle system. Also, copy + paste your programs to the text window on the test page.
- A few test cases against each problem is given for your references and but not limited to.
- There are three problems and the maximum time allowed is 120 minutes.
- **Do not use any function in this lab.**

1. **Consider an array of integers of size N. Do the following:**
   a. **Read n elements (n<=N) and store them in the array, where N = 100.**
   Use macro for setting the limit.
   (Hint: Set the upper limit using `#define N 100`**)**
   b. **Rearrange**
      i. **all negative numbers on the left side of the array and**
      ii. **all positive numbers on the right side of the array**

   Do not use any other array.
   The rearranging should not change the original order of the entered numbers.

Test cases: **[30]**

#	INPUT	OUTPUT
1	5 -1 5 6 -4 -3	Original Array: -1 5 6 -4 -3 Rearraged Array: -1 -4 -3 5 6
2	5 -1 -3 4 5 6	Original Array: -1 -3 4 5 6 Rearranged Array: -1 -3 4 5 6
3	6 -1 -2 -3 -4 -5 -6	Original Array: -1 -2 -3 -4 -5 -6 Rearranged Array: -1 -2 -3 -4 -5 -6
4	6 1 2 3 4 5 -6	Original Array: 1 2 3 4 5 -6 Rearranged Array: -6 1 2 3 4 5
5	4 1 4 7 23	Original Array: 1 4 7 23 Rearranged Array: 1 4 7 23
6	101	Error: n > 100



```c
// Code creator: Nishkal (nishkal@iitkgp.ac.in)
#include <stdio.h>
#define N 100   // Maximum number of elements in the array
int main()
{
    int n, arr[N]; // n is the number of elements in the array taken from user
    int i, j=0, temp;   // i is the index of the array, j is the index of the first positive number

    printf("\nEnter the number of elements (n <= %d): ", N);
    scanf("%d", &n);
    if (n > N) // Checking if the number of elements is greater than the maximum number of elements
        return printf("Error: n > %d", N); // If yes, then return an error message

    printf("\nEnter %d elements: ", n);

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

    printf("\nOriginal Array: ");
    for (i = 0; i < n; i++) // Printing the original array
        printf("%d ", arr[i]);

    for (i = 0; i < n; i++)
    {
        // If the element is negative,
        // then shift all the elements to the right of the first positive number by one position
        if (arr[i] < 0)
        {
            temp = arr[i];
            for (int k = i; k > j; k--) // Shifting all the elements to the right of the first positive number by one position
                arr[k] = arr[k - 1];
            arr[j] = temp;
            j++;   // Incrementing the index of the first positive number
        }
    }

    printf("\nRearranged Array: ");
    for (i = 0; i < n; i++) // Printing the rearranged array
        printf("%d ", arr[i]);

    return printf("\n");
}
```

2. Write a program to:
   a. Read n (<=N=100) numbers from the user.
   b. Store the numbers in an array.
   c. Remove all the duplicate entries.
   d. Print the array elements after the removal of all duplicate numbers.

   Do not use any other array.

   The duplicate removal should not change the original order of the entered numbers.

Test cases: **[30]**

#	INPUT	OUTPUT

1	8
	5 96 33 55 55 33 33 5
		Original Array: 5 96 33 55 55 33 33 5
		Duplicates:
		        5 (2)
		        33 (3)
		        55 (2)
		Unique Array: 5 96 33 55

2	5 1 2 2 3 3	Original Array: 1 2 2 3 3 Duplicates: 2 (2) 3 (2) Unique Array: 1 2 3
3	5 1 1 1 1 1	Original Array: 1 1 1 1 1 Duplicates: 1 (5) Unique Array: 1
4	5 0 0 1 -1 -1	Original Array: 0 0 1 -1 -1 Duplicates: 0 (2) -1 (2) Unique Array: 0 1 -1
5	5 1 2 3 4 5	Original Array: 1 2 3 4 5 Duplicates: No Duplicates Found Unique Array: 1 2 3 4 5
6	101	Error: n > 100

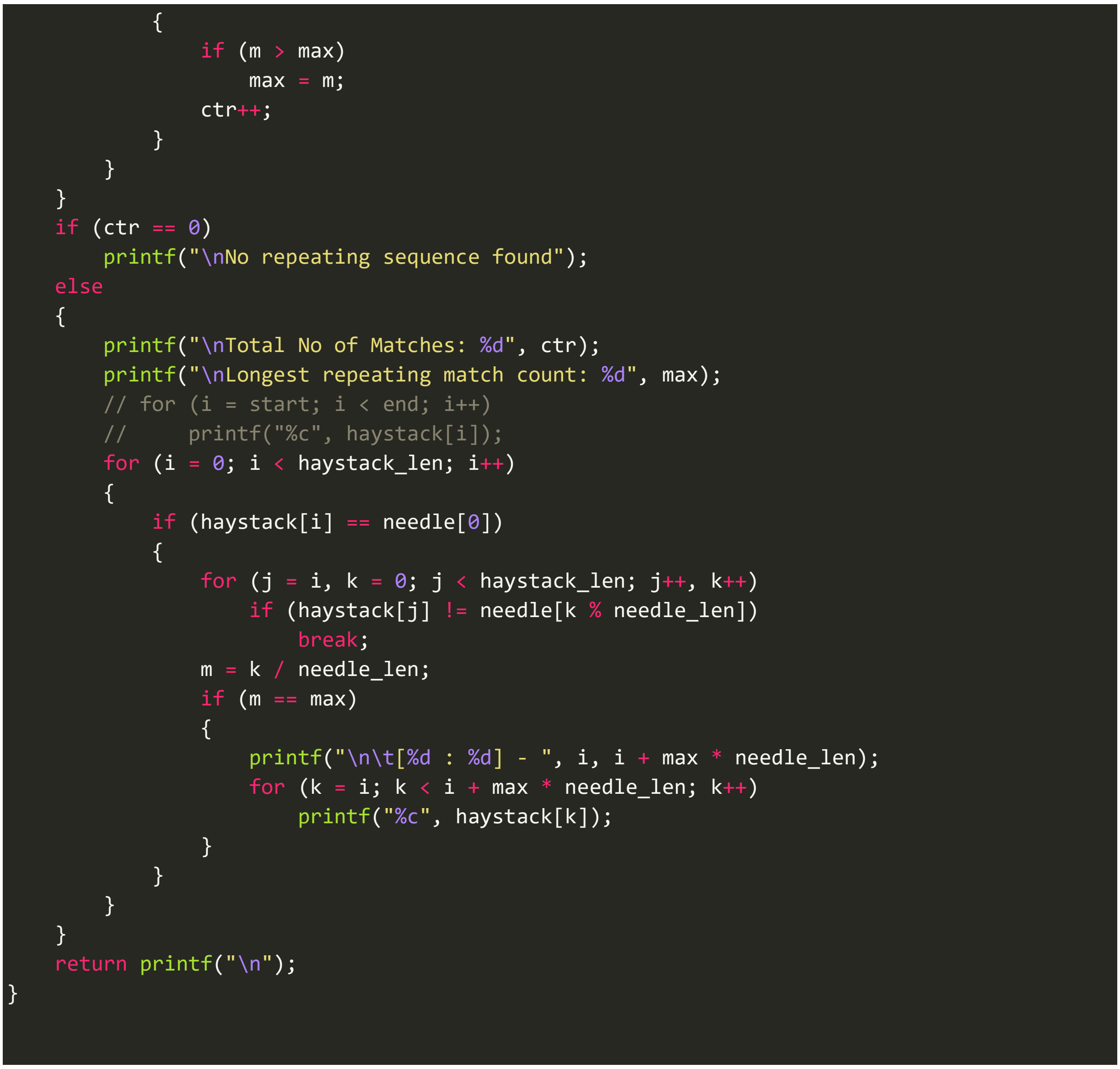
```c
// Code creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
#include <stdio.h>
#define N 100 // Maximum number of elements in the array

int main()
{
    int n, arr[N];         // n is the number of elements in the array taken from user
    int i, ctr = 0, j, k, flag=0; // i is the index of the array,

    printf("\nEnter the number of elements (n <= %d): ", N);
    scanf("%d", &n);
    if (n > N)                              // Checking if the number of elements is greater
than the maximum number of elements
        return printf("Error: n > %d", N); // If yes, then return an error message

    printf("\nEnter %d elements: ", n);
    for (i = 0; i < n; i++) // Reading the elements of the array
        scanf("%d", &arr[i]);

    printf("\nOriginal Array: ");
    for (i = 0; i < n; i++) // Printing the original array
        printf("%d ", arr[i]);

    // Printing all the duplicate elements with their Frequency
    printf("\nDuplicates: ");
    for (i = 0; i < n; i++)
    {
        ctr = 1; // Frequency of the duplicate element
        for (j = i + 1; j < n; j++)
        {
            if (arr[i] == arr[j])
            {
                ctr++; // Incrementing the frequency of the duplicate element
                // Shift all elements to the left by one position and decrease the size of
the array by one
                for (k = j; k < n - 1; k++)
                    arr[k] = arr[k + 1];
                n--; // Decreasing the size of the array by one
                j--; // Decrementing the index of the array
            }
        }
        if (ctr > 1) // If the frequency of the duplicate element is greater than 1, then
print it
        {
            printf("\n\t%d (%d)", arr[i], ctr);
            flag=1;
        }
    }
    if(flag==0)
        printf("\n\tNo Duplicates Found");

    printf("\nUnique Array: ");
    for (i = 0; i < n; i++) // Printing the unique array
        printf("%d ", arr[i]);

    return printf("\n");
}
```

3. Write a program to:
    a. Define an array which will store only 1's and 0's. (Hint: use char array)
    b. Read the binary pattern of length n (<=N=100), call it Haystack.
    c. Read another small sequence of binary pattern of length m (<=M=10), call it Needle.
    d. Scan the input array to find the longest repeating sequence of the needle and print it along with the start index and end index.
    If multiple longest sequence is present print all of them.

Test cases: **[40]**

#	INPUT	OUTPUT
1	01100110101101010101100101010110 1010	Haystack: 01100110101101010101100101010110 Needle: 1010 Total No of Matches: 6 Longest repeating match count: 2 [11 : 19] - 10101010
2	01100110101101010101100101010110 01	Haystack: 01100110101101010101100101010110 Needle: 01 Total No of Matches: 12 Longest repeating match count: 4 [12 : 20] - 01010101 [22 : 30] - 01010101
3	1010110011001100 1110	Haystack: 1010110011001100 Needle: 1110 No repeating sequence found
4	101010101010101010101 101	Haystack: 101010101010101010101 Needle: 101 Total No of Matches: 10 Longest repeating match count: 1 [0 : 3] - 101 [2 : 5] - 101 [4 : 7] - 101 [6 : 9] - 101 [8 : 11] - 101 [10 : 13] - 101 [12 : 15] - 101 [14 : 17] - 101 [16 : 19] - 101 [18 : 21] - 101
5	111111111111111111 1	Haystack: 111111111111111111 Needle: 1 Total No of Matches: 18 Longest repeating match count: 18 [0 : 18] - 111111111111111111
6	1001 1001	Haystack: 1001 Needle: 1001 Total No of Matches: 1 Longest repeating match count: 1 [0 : 4] - 1001
7	1111 10101	Haystack: 1111 Needle: 10101 Error: needle_len > haystack_len
8	10102101011	Error: Invalid character in the haystack

```c
// Code creator: Nishkal Prakash (nishkal@iitkgp.ac.in)
#include <stdio.h>
#define N 100 // Maximum number of elements in the array
#define M 10  // Maximum number of elements in the array

int main()
{
    char haystack[N]; // Haystack is the array in which we search for the needle
    char needle[M];   // Needle is the small pattern which we search for in the haystack
    char c;
    int haystack_len = 0, needle_len = 0; // haystack_len stores the length of the
haystack, needle_len stores the length of the needle
    int i, j, k, ctr = 0;                 // i,j,k are loop variables, ctr is the number of
times the needle is repeated in the haystack
    int m = 0, max = 0;                   // m stores the current number of times the
needle is repeated in the haystack
    // Read the Haystack
    while ((c = getchar()) != '\n')
    {
        if (c == '0' || c == '1')
            haystack[haystack_len++] = c;
        else
            return printf("\nError: Invalid character in the haystack");
    }

    // Read the Needle
    while ((c = getchar()) != '\n')
    {
        if (c == '0' || c == '1')
            needle[needle_len++] = c;
        else
            return printf("\nError: Invalid character in the needle");
    }

    // Print the Haystack
    printf("\nHaystack: ");
    for (i = 0; i < haystack_len; i++)
        printf("%c", haystack[i]);

    // Print the Needle
    printf("\nNeedle: ");
    for (i = 0; i < needle_len; i++)
        printf("%c", needle[i]);

    // Error checking
    if (needle_len > haystack_len)
        return printf("\nError: needle_len > haystack_len");

    // Search for the longest repeating sequence of the small pattern
    for (i = 0; i < haystack_len; i++)
    {
        if (haystack[i] == needle[0])
        {
            for (j = i, k = 0; j < haystack_len; j++, k++)
                if (haystack[j] != needle[k % needle_len])
                    break;
            m = k / needle_len;
            if (m != 0)
```



```
            {
                if (m > max)
                    max = m;
                ctr++;
            }
        }
    }
    if (ctr == 0)
        printf("\nNo repeating sequence found");
    else
    {
        printf("\nTotal No of Matches: %d", ctr);
        printf("\nLongest repeating match count: %d", max);
        // for (i = start; i < end; i++)
        //     printf("%c", haystack[i]);
        for (i = 0; i < haystack_len; i++)
        {
            if (haystack[i] == needle[0])
            {
                for (j = i, k = 0; j < haystack_len; j++, k++)
                    if (haystack[j] != needle[k % needle_len])
                        break;
                m = k / needle_len;
                if (m == max)
                {
                    printf("\n\t[%d : %d] - ", i, i + max * needle_len);
                    for (k = i; k < i + max * needle_len; k++)
                        printf("%c", haystack[k]);
                }
            }
        }
    }
    return printf("\n");
}
```

---*---