LAB ASSIGNMENT 9

1. Given a string, find its first non-repeating character.

Input: arunkumar

Output: n

```
#include <stdio.h>

#include <string.h>


int main(void) {


 char str[100] = "applea";

 int len  = strlen(str);

 int flag;


 /* Two loops to compare each

    character with other character

    */

 for(int i = 0; i < len; i++) {


     flag = 0;


     for(int j = 0; j < len; j++) {


        /* If it's equal and indexes
        is not same */
      if((str[i] == str[j]) && (i != j)) {
          flag = 1;
          break;
      }
    }

     if (flag == 0) {
        printf("First non-repeating character is %c",str[i]);
        break;
     }

 }

 if (flag == 1) {
    printf("Didn't find any non-repeating character");
 }

  return 0;
}
```

2. Write a program to print all permutations of a given string.

Input: ABC

Output: ABC ACB BAC BCA CBA CAB

```
#include <stdio.h>

#include <string.h>


void changePosition(char *ch1, char *ch2)

{

    char tmp;

    tmp = *ch1;

    *ch1 = *ch2;

    *ch2 = tmp;

}

void charPermu(char *cht, int stno, int endno)

{

   int i;

   if (stno == endno)

     printf("%s  ", cht);

   else

   {

       for (i = stno; i <= endno; i++)

       {

          changePosition((cht+stno), (cht+i));

          charPermu(cht, stno+1, endno);
```

```
         changePosition((cht+stno), (cht+i));
      }
   }
}

int main()
{
   char str[] = "abcd";
   int n = strlen(str);
   printf(" The permutations of the string are : \n");
   charPermu(str, 0, n-1);
    printf("\n\n");
   return 0;
}
```

3. Recursively removes all adjacent duplicates.

Input: abccbd

Output: ay

First “abccbd” is reduced to “abbd”.

The string “abbd” contains duplicates,

so it is further reduced to “ad”.

```
#include <stdio.h>
```

```
#include <string.h>

// Function to remove all adjacent duplicates from the given string
char* removeAdjDup(char* str, int n)
{
    // base case
    if (n == 0) {
        return str;
    }

    // `k` maintains the index of the next free location in the result,
    //and `i` maintains the current index of the string
    int i, k = 0;
    int len = strlen(str);

    // start from the second character
    for (i = 1; i < len; i++)
    {
        // if the current character is not the same as the
        // previous character, add it to the result
        if (str[i - 1] != str[i]) {
            str[k++] = str[i - 1];
        }
        else {
            // remove adjacent duplicates
            while (i < len && str[i - 1] == str[i]) {
                i++;
            }
        }
    }

    // add the last character to the result
    str[k++] = str[i - 1];

    // null terminate the string
    str[k] = '\0';

    // start again if any duplicate is removed
    if (k != n) {
        return removeAdjDup(str, k);
    }

    // if the algorithm didn't change the input string, that means
    // all the adjacent duplicates are removed
    return str;
}

int main(void)
```

```
{
   char str[] = "DBAABDAB";
   int n = strlen(str);

   printf("The string left after the removal of all adjacent duplicates is
%s",
      removeAdjDup(str, n));

   return 0;
}
```