STRINGS

Name | ADDRESS (TEXT DATA) → [Array of characters]

1. char constant → 'A'
2. char string → "PASS"

Input of string :-

1. char res[5] = "PASS"

P	A	S	S	\0

2. char res[5];
   printf("Enter your file name: \n");
   gets(res);

output of a string :-

1. printf("file name: %s", res);
2. Puts(res);
3. for(i=0; res[i] != '\0'; i++)
   {
   printf("%c", res[i]);
   }

Using loops ←

Here, res → string variable.

* **Finding Length of a String** :- Already done in C Language notes.

```
int main()
{
    int i, l = 0;
    char name[30];
    printf("Enter your name : \n");
    gets(name);
    for(i=0 ; name[i] != '\0' ; i++)
    {
        l++;
    }
    printf("length : %d \n", l);
}
```

Changing Case of a String :- Lower Case → Upper Case
Upper Case → Lower Case

A-Z
65-90

a-z
97-122

Also, It is already done in C Language notes.

```
int main()
{
    int i;
    char s[40];
    printf("Enter string : \n");
    gets(s);
    for(i=0; s[i]!='\0'; i++)
    {
        if(s[i]>=65 && s[i]<=90)
            s[i] = s[i]+32;
        else if(s[i]>=97 && s[i]<=122)
            s[i] = s[i]-32;
    }
    printf("%s", s);
}
```

★ Counting Words & Vowels in a String :- It's also already done in C Language notes.

- For Counting Vowels :-

```
int main ()
{
   int i, v=0;
   char name[30];
   printf ("Enter your name : \n");
   gets (name);
  for (i=0 ; name[i] != '\0' ; i++)
     {
        Switch (name[i])
           {
             Case 'A' : Case 'E' : Case 'I' : Case 'O' ;
             Case 'U' : Case 'a' : Case 'i' : Case 'U
             Case 'e' : Case 'O' : v++ ;
                        break;
           }
   printf (" In Vowels : %d", v);
}
```

Counting Words :-

```
int main()
{
   int i, v=1;
   char name[30];
   printf("Enter string : \n");
   gets(name);
   for(i=0 ; name[i] != '\0' ; i++)
      {
         if (name[i] == '_' && name[i+1] != '_')
               v++;
      }
   printf("no. of words : %d", v);
}
```

Validating a String

:- Is this string valid or not. like when we enter a password / username it says that it contains only letters / no's / special symbols In these cases validation of string is used

```
int main ( )
{
  int i;
  Char name [40];
  printf ("Enter String: \n");
  gets (name);
  for (i=0 ; name [i] != '\0' ; i++)
    {
      if ((!(name [i] >= 65 && name [i] <= 90) &&
          ! (name [i] >= 97 && name [i] <= 122) &&
          ! (name [i] >= 48 && name [i] <= 57))
            {
              Prince = 1;
      else      } break;
          Prince = 0;
    }
  if (Prince == 0)
    printf (" Valid string");
  else if (Prince == 1)
    printf (" Not valid");
}
```

Comparison of Two Strings :-

Compare two strings by checking their elements step by step.

```
int main()
{
    int i, j;
    char A[30], B[30];
    printf("Enter string A: \n");
    gets(A);
    printf("Enter string B : \n");
    gets(B);
    for(i=0, j=0; A[i] != '\0' && B[i] != '\0'; i++, j++)
    {
        if (A[i] != B[j])
        {
            printf("strings doesnt match");
        } break;
    }
    if (A[i] == B[j])
        printf("equal");
    else if (A[i] < B[j])
        printf("string A comes first in Dictionary");
    else if (A[i] > B[j])
        printf("string B comes first in Dictionary");
}
```

★ Checking Palindrome

:- Palindrome is what when we reverse the string if it is same then it is Palindrome

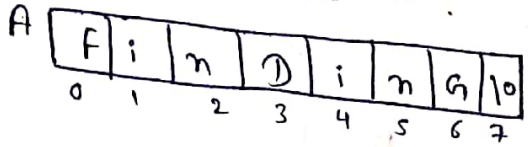
```
int main()
{
  int i, l=0, prince;
  char A[30];
printf("Enter string A: \n");
gets(A);
for(i=0; A[i]!='\0'; i++)
   { l++;
   }
for(i=0 ; i<l/2; i++)
  { if (A[i] != A[l-1])
      {
        printf ("Not a Palindrome");
        prince = 1;
        break;
      }
    else
      prince = 0;
      l--;
  }
    if (prince == 0)
     printf ("It is palindrome");
}
```

Duplicates in a String :-

Finding duplicated elements or characters in a string

1st Method :-

Comparing with others.

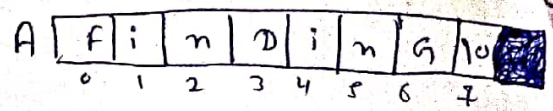



```
int main()
{
    int i, count, j;
    char A[30];
    printf("Enter string: \n");
    gets(A);
    for(i=0; A[i] != '\0'; i++)
    {
        count = 1;
        if(A[i] != -1)
        {
            for(j=i+1; A[j] != '\0'; j++)
            {
                if(A[i] == A[j])
                {
                    count++;
                    A[j] = -1
                }
            }
            if(count > 1)
            {
                printf("Duplicated character : %c\n", A[i]);
            }
        }
    }
}
```

Time Complexity

$O(n^2)$

2nd Method :- Using Hash table [for both upper & lower case]




```
int main()
{
    int i
    char A[30];
    printf("Enter string : \n");
    gets(A);
    int H[52];
    for(i=0; i<52; i++)
        H[i]=0;
    for(i=0; A[i]!='\0'; i++)
    {
        if(A[i]>=65 && A[i]<=90)
            H[A[i]-65]++;
        else if(A[i]>=97 && A[i]<=122)
            H[A[i]-97+26]++;
    }
    for(i=0; i<52; i++)
    {
        if(H[i]>1 && i<=25)
        {
            printf("Duplicated character: %c\n", i+65);
            printf("%c appearing %d times\n", i+65, H[i]);
        }
        else if(H[i]>1 && (i>25 && i<=51))
        {
            printf("Duplicated character: %c\n", i+97-26);
            printf("%c appearing %d times\n", i+97-26, H[i]);
        }
    }
```

Time Complexity
O(n)

Method :- Using Bits [This method can also be used for Arrays but it is more useful for strings]

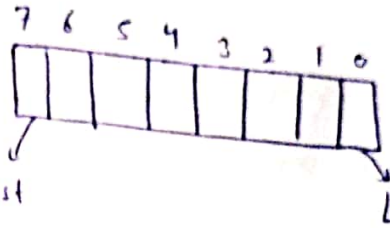
Concepts :-

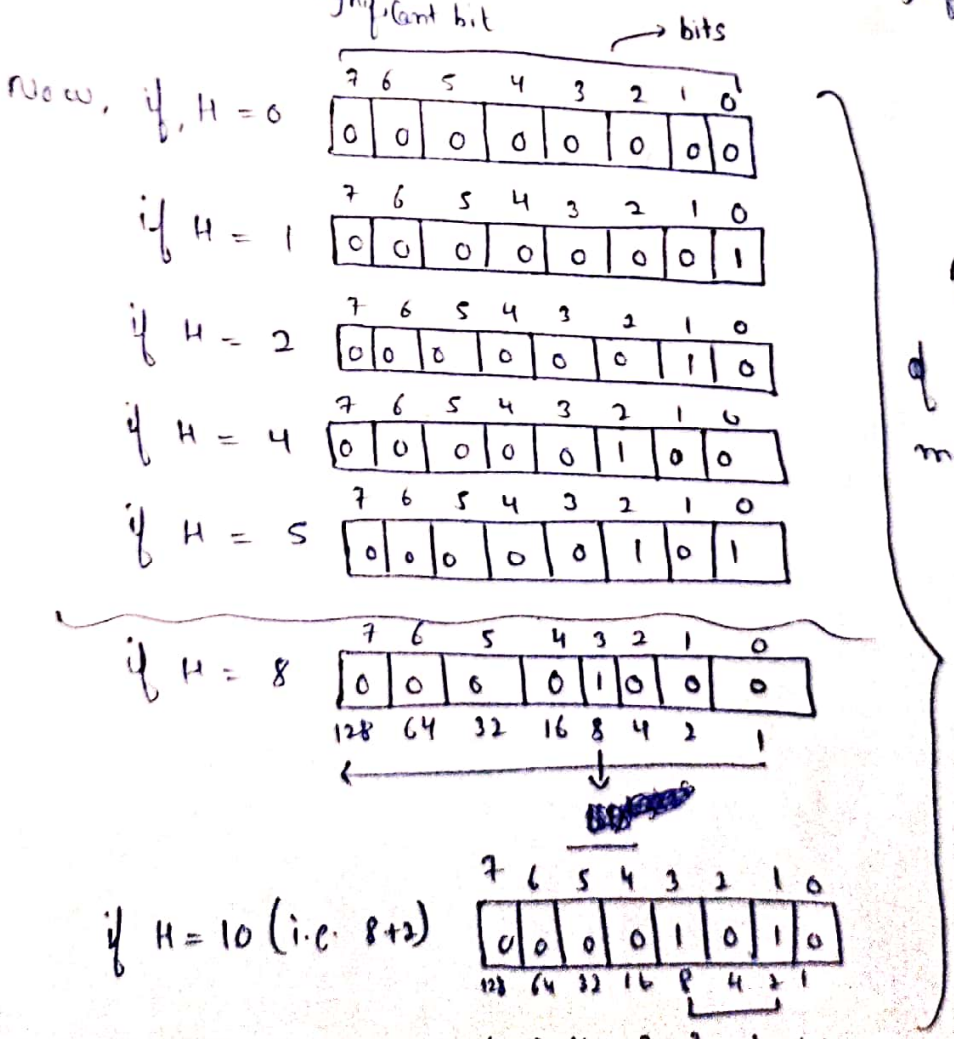
Bitwise Operations :-

1. Left shift <<
2. Bits ORing (Merging)
3. Bits ANDing (masking)

1 Byte → 8 bits

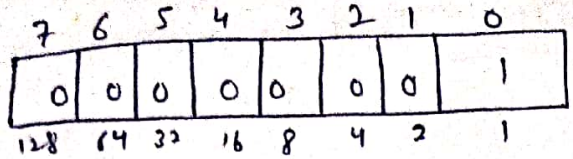
Eg :- 1 Byte → 'H'





Binary form of no. stored in memory in bits.

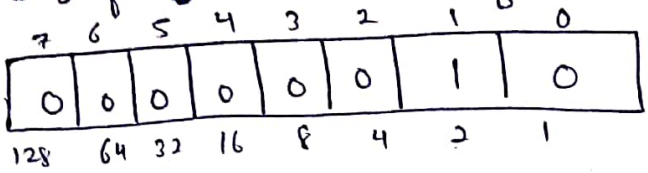
(1) Shift operation :-



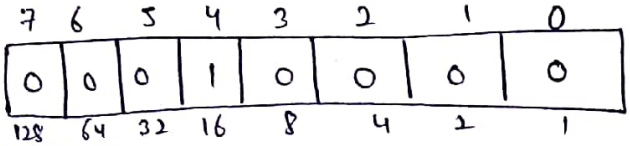
H = 1

H = H<<1 → Results in H [H will be modified]

It means all bits shift by one place to left Side

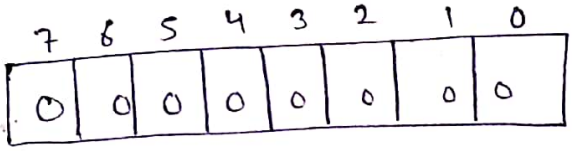


For, H = H<<4



H will be 16 Now

(2) Bit ANDing :-



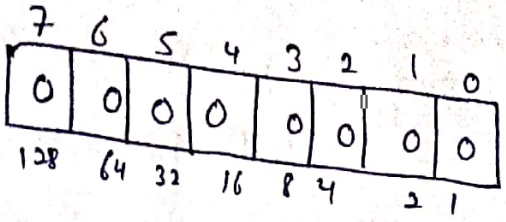
a & b	
1 & 1	= 1
1 & 0	= 0
0 & 1	= 0
0 & 0	= 0

a = 10 → 1010

b = 6 → 0110

a & b → 0010 = binary form of 2

ORing :-



a	b	a | b
1	1	1
0	0	0
1	0	1
0	1	1

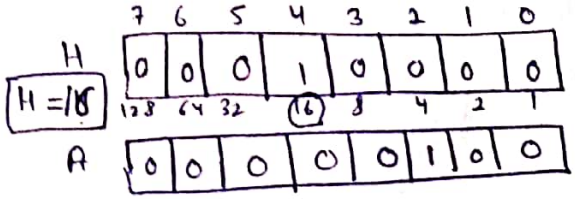
a = 10 → 1010

b = 6 → 0110

a | b → 1110 → binary form of 14

★ **Now, understand Masking** :- In this we have to know that whether the bit is on/off, so to check this we take another variable assigned as 1 then we will shifting and by doing ANDing operation, if we get 1, then bit is (i.e. True) on, other wise if we get 0, then bit will off.

Showing bit is off ⟹



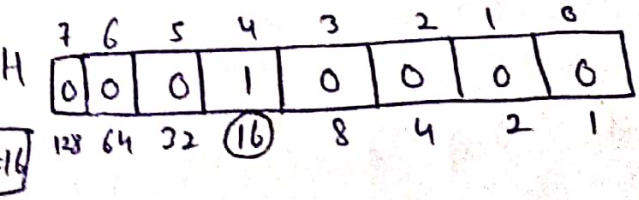
H = 16

H = A & H → operation

A = 1

A = A << 2

Showing bit is on ⇒



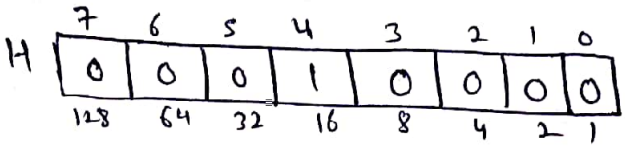
H = 16

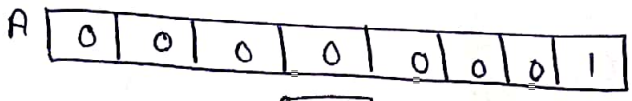
H = A & H → operation

A = 1

A = A << 4

☆ **Now Understand Merging** :- In this we have set that the bit will have to be on | off. In this we will use ORing operation in such a way that we will take another variable and then shifting (perform) and then we apply ORing b|w A & H, so to set that bit as on | off.

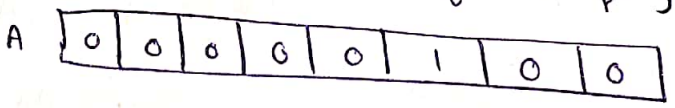




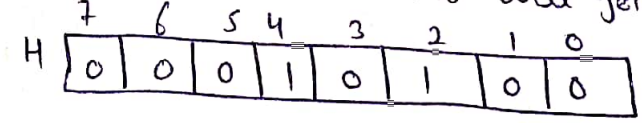
`A = 1`

`A = A << 2` (Suppose we have set bit-2 on)

And then A will becomes After shifting



Now, Perform ORing and we will get bit-2 on (H=A | H)



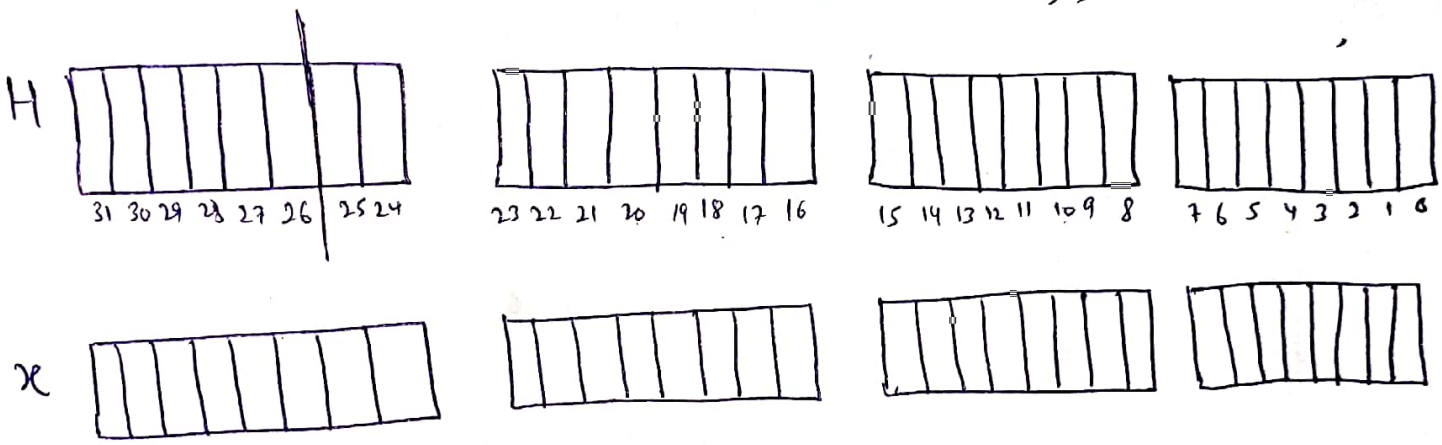
Imp Note :-

- If we want to check whether the bit is on | off then we will perform Masking.
- If we want to set the bit as 'on', then we will perform Merging.

6;t

Code :- [It is similar to hash table]. For lower case only

Now, for this, we require 26 bits as we require 26 size in hash table, but we have to take 32 bits as common and the remaining ones will be untouched, So, Now we also require another variable for performing masking and merging (take as $x = 0$);



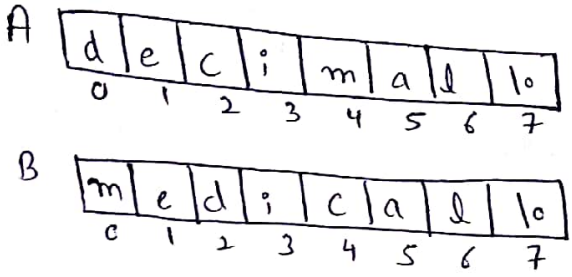
Procedure :- (i) Pehle 'x' ko set krongh as '1' then shift it to the required bit (i.e. A[i]-97) then we will check whether the bit is on/off, if it is on, then we will print it as duplicated.

(ii) Else, we will perform merging to set this bit as on, so that Jab next time same character Aaye toh, it will enter into masking condition and print it as duplicated.

Program - Code :-

```
int main()
{
    int i, H=0, x=0;
    Char A[20];
    printf("Enter string : \n");
    gets(A);
    for(i=0; A[i]!='\0'; i++)
    {
        x=1;
        x = x << (A[i]-97);
        if((x & H)>0)  || masking Condition
        {
          printf("Duplicated character: %c", A[i]);
        }
        else
          H = x | H;   || Merging Condition
    }
}
```

checking if two strings are Anagram :- if two strings are formed using same set of alphabets then strings are said to be Anagram.



1st Method :- By Comparison of first element of String A with all elements of String B but it will take time $O(n^2)$ So, we will use Hash table

2nd Method :- Using Hash Table [For both upper and lower case]

```
int main()
{
int i
Char A[30], B[30];
printf(" Enter String A : \n");
gets(A);
printf(" Enter String B : \n");
gets(B);
```

Time Complexity
O(n)

```
int H[52];
for(i=0;i<52;i++)
    H[i]=0;
for(i=0; A[i]!='\0';i++)
  {
    if (A[i]>=97 && A[i]<=122)
        H[A[i]-97]++;
    else if (A[i]>=65 && A[i]<=90)
        H[A[i]-65+26]++;
  }
for(i=0; B[i]!='\0';i++)
  {
    if (B[i]>=97 && B[i]<=122)
      {
        H[B[i]-97]--;
        if (H[B[i]-97]<0)
          {
            printf("Not Anagram\n");
          }break;
      }
    else if (B[i]>=65 && B[i]<=90)
      {
        H[B[i]-65+26]--;
        if (H[B[i]-65+26]<0)
          {printf("Not Anagram\n");
          }break;
      }
  }
  if (B[i]=='\0')
      printf("Is Anagram");
//main closed <-- }
```

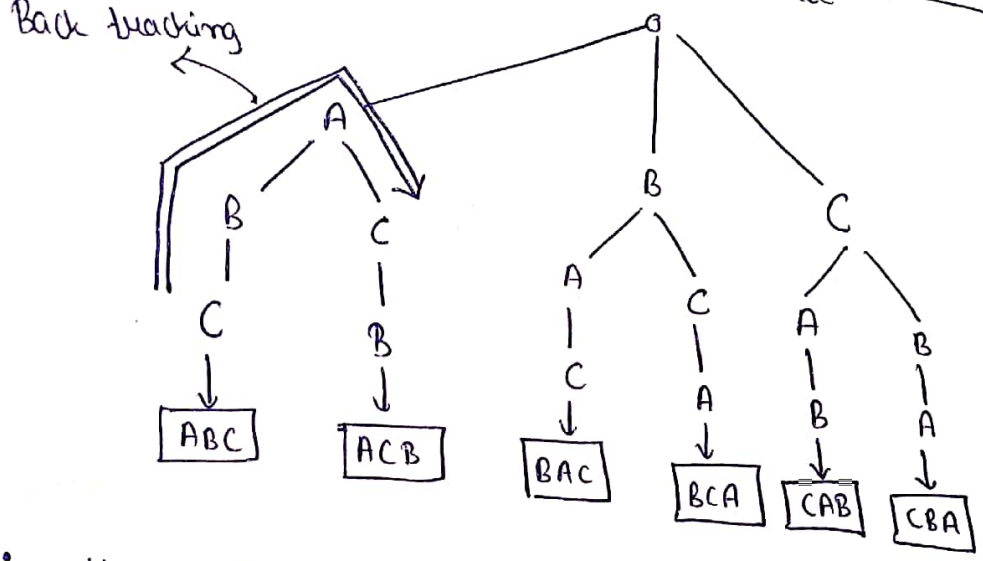
Permutation of String :- Permutation means all Arrangements

Eg :- S | A | B | C | \0 |
0 1 2 3

Permutation = n!

1. ABC
2. ACB
3. BAC
4. BCA
5. CAB
6. CBA

State Space tree → It is a tree in which result is shown in form of leaves.



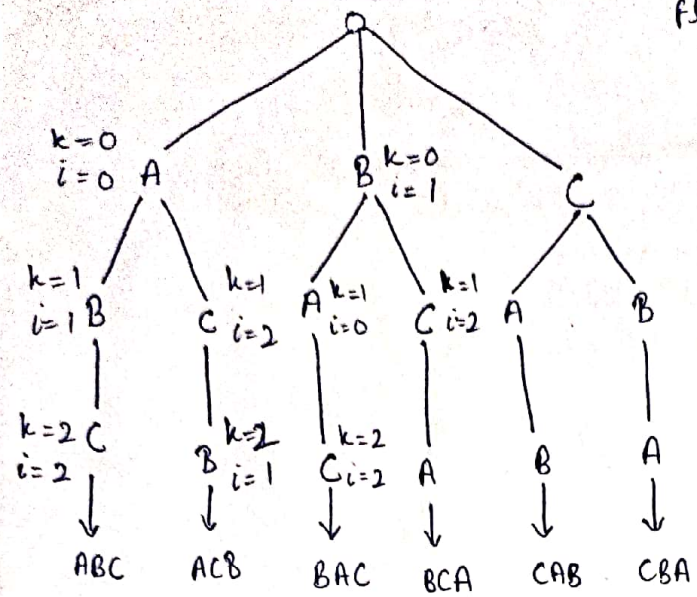
→ Leaf node.

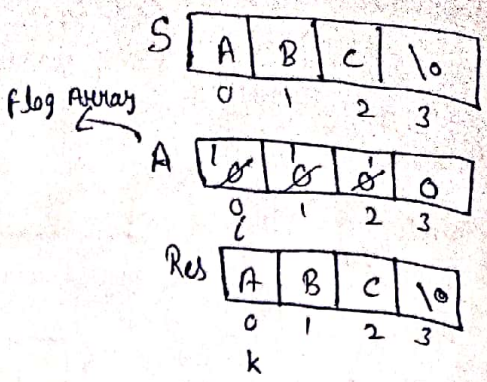
- Here, finding out all the permutations is known as Brute force.
- Here we are use back tracking, that is when we find one combination then go back and find New route and this can be implamanted by using Recursion. So we are using Recursion to perform Back tracking and then achieve Brute force.

RECURSION → BACK TRACKING → BRUTE FORCE

- We can do this using loops also, but mostly a favourable method is Recursion for back tracking to achieve Brute force.

State Space tree





Tracing :-

★ In first call, we will check whether in A, its zero or not, yes it is zero, so make it as 1 and write 'A' in Res[0] and make it call Again recursively

★ Now, k = 1 & again start 'i' from 0. check its free space Availaible ?. No, so increment i, check here; yes its availaible, so mark this as 1 & write 'B' in Res[1] and Call Again.

★ Now, k = 2, again start 'i' from 0; but at A[2] space is free, so make it 1 and write 'C' in Res[2]. Call itself again, but Now no need to scan array bcz it reaches '\0'. it means we got '\0', so end this string & print the result in next call, when k = 3.

★ Once we printed 'ABC', go back to previous call, Here, k = 2, i = 2, Continue Scanning making it as zero (i.e A[2]) then again go back, Continue scanning but make zero again (i.e A[1]). Now on Continue scanning, so, on k=1, i=2

make it as 1 & write Res[i] as 'c'. The again calling and start scanning, so, initially k=0, i=0, bcz when we call forward, start scanning from 0, so, at k = 1 & i = 1 space is free, so, Copy 'B' at Res[1] & increment A[i]. then when we call again, k = 3, so print the result and again going back and proceed as what we discussed before

- So, this is recursive (back tracking).

Code :-

```
int main()
{
  int j;
  char A[30];
  printf("Enter String: \n");
  gets(A);
  void Perm(char A[], int k);
    Perm(A, 0);
}
Void Perm(char A[], int k)
{
  static int a[10] = {0};
  static char Res[10];
  int i;
  if (A[k] == '\0')
    {
      Res[k] = '\0';
      puts(Res);
    }
  else {
    for(i=0; A[i] != '\0'; i++)
      { if (a[i] == 0)
        {
          Res[k] = A[i];
          a[i] = 1;
          Perm(A, k+1);
          a[i] = 0;
        }
      }
  }
}  // function closed
```