Notebook
Pg. No.
Date / /

Section 8

Strings

Notebook
Pg. No.
Date / /

Introduction to Strings -

1. Character set / ASCII Codes
2. Character Array
3. String
4. Creating a String

UNICODES (Hindi ... etc)

- **Character set / ASCII Codes** - (English)

ASCII, abbreviated from American standard code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications equipment, and other devices.

A → 65
B → 66
C → 67
⋮
Z → 90

Numerical Symbol

a → 97
b → 98
c → 99
⋮
z → 122

0 → 48
1 → 49
2 → 50
⋮
9 → 57

↵ (Enter) → 10

Space → 13

Esc → 27

Ascii codes Range → (0-127) → 7 bits.

total → 128

$2^7 = 128$

Unicodes → (For All Language)

↳ 2 bytes of memory

↳ 16 bits

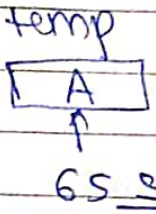
Represent in 4×4 bits

Hexadecimal Code

Notebook
Pg. No.
Date / /

char temp; ← 1 byte

temp = 'A'; ✓



temp = 'AB'; X

temp = A ; X

temp = "A" ; X

cout << temp; → A

printf("%c", temp); → A

printf("%d", temp); → 65

~~cout<<~~

Array of Characters :-

char X[5];

char X[5] = {'A', 'B', 'C', 'D', 'E'};

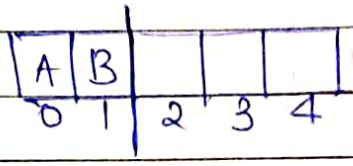
A	B	C	D	E
0	1	2	3	4

X[ ] = {'A', 'B', 'C', 'D', 'E'}

char X[5] = {65, 66, 67, 68, 69}

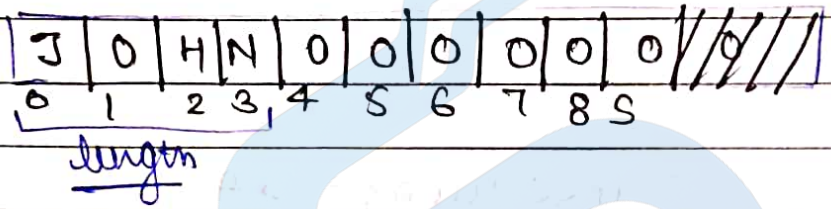
Notebook
Pg. No.
Date / /

char X[5] = {'A', 'B'}



char name[10] = {'J', 'O', 'H', 'N'};

Array of characters



'\0' → Null character
→ String delimiter
→ End of string char
→ String Terminator

char name[] = {'J', 'O', 'H', 'N', '\0'}

Size of Array = 5

char name[] = "John";

printf("%s", name); → John

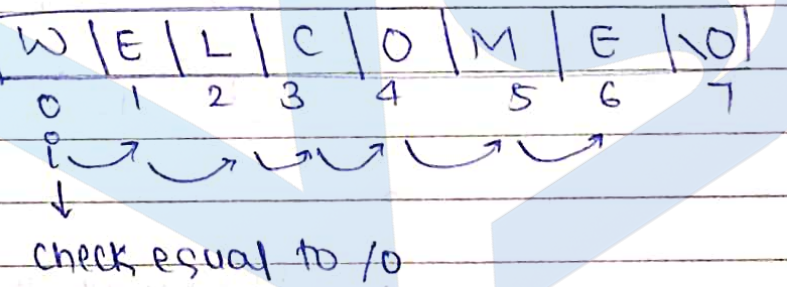
scanf("%s", name);

↳ Useful for only one word

gets(name)

↳ useful for more than one word

- Finding length of string




```
int main()
{
    char s[] = "welcome";
    int i;
    for(i=0; s[i]!='\0'; i++)
    {

    }
    printf("length is %d", i);
    return 0;
}
```

Changing case of a string

→ Uppercase to lowercase

A	W	E	L	C	O	M	E	\0
	0	1	2	3	4	5	6	7

A – 65
B – 66
⋮
Z → 90

a → 97
b → 98
⋮
z = 122

97 – 65 = 32
98 – 66 = 32
⋮ ⋮ ⋮

```
int main()
{
    char A[] = "WELCOME";
    int i;
    for(i=0; A[i]!='\0'; i++)
    {
        A[i] = A[i]+32;
    }
    printf("%s", A)
}
```

Lowercase to uppercase

```
int main()
{
    char B[] = "welcome";
    int i;
    for(i=0; B[i]!='\0'; i++)
    {
        B[i] = B[i]-32;
    }
    cout<<B;
}
```

Toggling case → Upper to lower and lower to upper

A	w	E	l	C	o	m	e	\0
	0	1	2	3	4	5	6	7

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

Counting words and vowels in a string -

↳ counting vowels and consonents

H	O	W		a	r	e		Y	O	U		\0

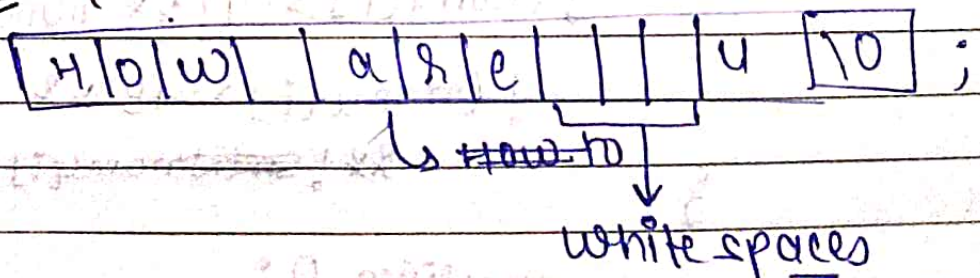
```
int main()
{
    char A[] = "How are you";
    int i;
    int vcount = 0;
    int ccount = 0;

    for(i=0; A[i] != '\0'; i++)
    {
        if(A[i]=='a' || A[i]=='e' || A[i]=='i' ||
           A[i]=='o' || A[i]=='u' || A[i]=='A'
           || A[i]=='E' || A[i]=='I' || A[i]='O'
           || A[i]=='U')
        {
            vcount++;
        }
    }

    cout<< "No. of vowels :=" << vcount;
    else if((A[i]>=65 && A[i]<=90) ||
            (A[i]>=97 && A[i]<=122))
    {
        ccount++;
    }

}
```

```
cout<<"Vowel:"<<vcount;
cout<<"Consonents:"<<ccount;
}
```

Counting number of words:-

```
int main()
{
    char A[] = "How are you";
    int i, word=0;
    for(i=0; A[i]!='\0'; i++)
    {
        if(A[i]==' ' && A[i-1]!=' ')
        {
            word++;
        }
    }
    cout<<"Number of words:"<<word+1;
}
```

Solution of this problem



Validating a String :-

name

A	n	i	l	3	2	1	\0
0	1	2	3	4	5	6	7

Valid String

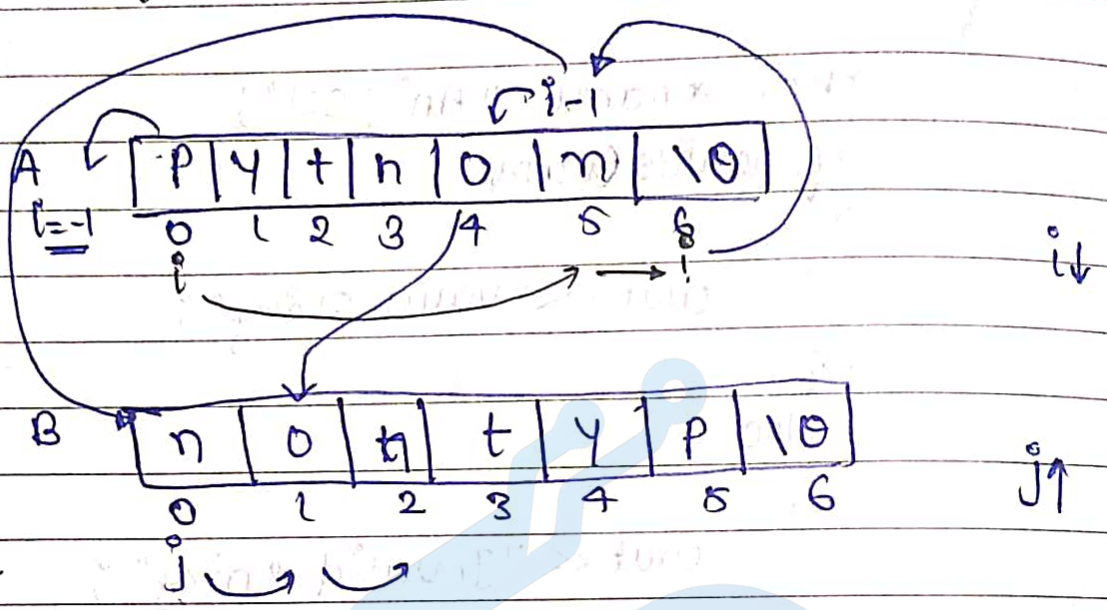
A	n	i	?	3	2	1	\0

Not Valid String

```
~~int main()~~
~~{~~
int valid(char *name)
{
    int i;
    for(i=0; name[i]!='\0'; i++)
    {
        ~~if((name[i]>=65 && name<=95)~~
        if(!(name[i]>=65 && name[i]<=90) &&
            !(name[i]>=97 && name[i]<=122) &&
              !(name[i]>=48 && name[i]<=57)
                && !name[i]=='_')
        {
            return 0;
        }
    }
    return 1;
}
```

```
int main()
{
    char *name = "Ani? 321";
    if (validatename))
    {
        cout << "valid string";
    }
    else
    {
        cout << "Invalid string";
    }
}
```

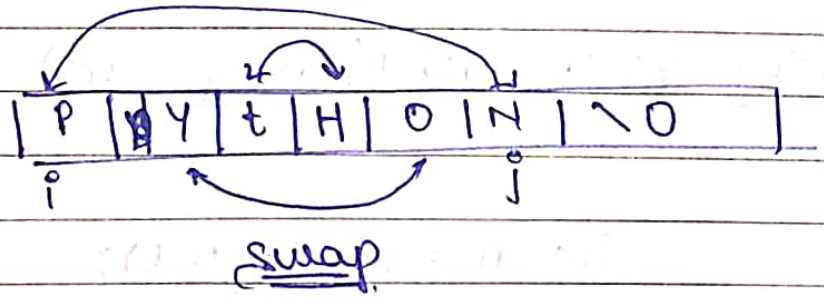
Reversing a string



Method 1:-

```
int main()
{
    char A[] = "python";
    char B[7];
    int i;
    for(i=0; A[i]!='\0'; i++)
    {
        XXXXXX
    }
    i=i-1;
    for(j=0; i>=0; i--, j++)
    {
        B[j] = A[i];
    }
    B[j] = '\0';

    cout << B;
}
```

Method 2:-



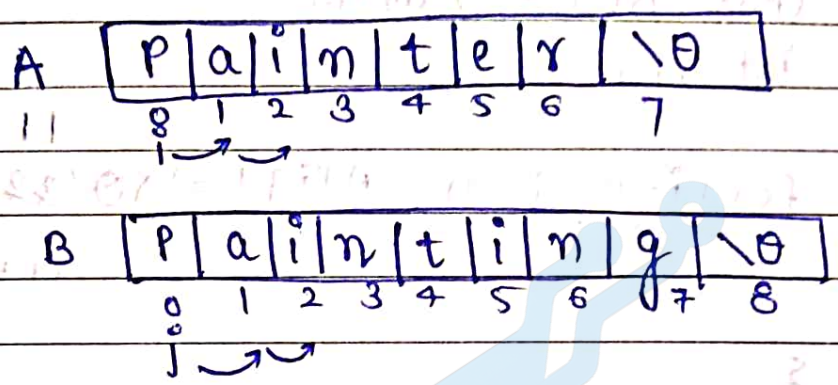

```
int main()
{
    char A[] = "python";
    char t;
    int i, j;

    for(j = 0 ; A[j] != '\0' ; j++)
    {

    }
    j = j-1;

    for( i=0; i<j ; i++, j--)
    {
        t = A[i];
        A[i] = A[j];
        A[j] = t;
    }
    cout << A;
}
```

Pg. No.
Date / /

```
* String reverseWord(string str)
  {

        int n = str.length();
        for(int i=0; i<n/2; i++)
        {
            swap(str[i], str[n-i-1]);
        }

        return str;

  }
```

~~Comparison~~

Comparing Strings and Checking Palindrome



A[i] == A[j] → checking (Move a head) if they are same

A[i] != A[j] → (No need to continue afterward)

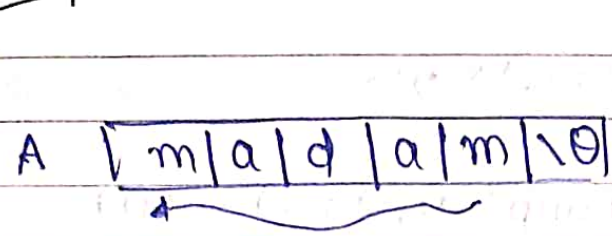
ASCII_codes(e) < ASCII_codes(i)

So, A is smaller than B.

Stop → Mismatch found
→ end of the string (both)

```
int main()
{
    char A[] = "Painter";
    char B[] = "Painting";
    int i, j;

    for(i=0, j=0; A[i] != '\0' || B[j] != '\0';
                                  i++, j++)

    {
        if(A[i] != B[j])
        {
            break;
        }
    }
    if(A[i] == B[j])
    {
        cout << "equal";
    }
    else if(A[i] < B[j])
    {
        cout << "smaller";
    }
    else
    {
        cout << "greater";
    }
}
```

Palindrome



① Reverse
② compare

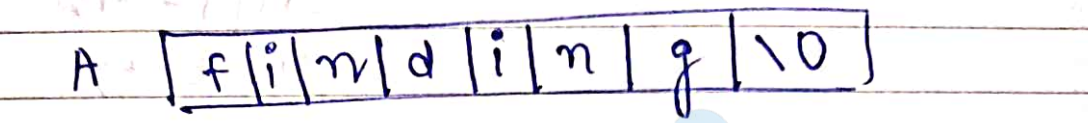
Palindrome String
↳ iri
↳ Nitin
↳ Nayan
↳ Azza

```
int main()
{
    char A[] = "madam";
    char B[6];
    char str[100], str2[100];
    cout << "Enter a string : ";
    gets(str);
    int i, j;
    for (i=0; str[i] != '\0'; i++)
    {

    }
    i = i-1;
    for( j=0; i>=0; i--, j++)
    {
        str2[j] = str[i];

    }
```

```
str2[j] = '\0';

if (strcmp(str, str2) == 0)
{
        cout<<" palindrome ";

}
else
{
        cout<<" Not palindrome";

}

return 0;

}
```

Notebook
Pg. No.
Date / /

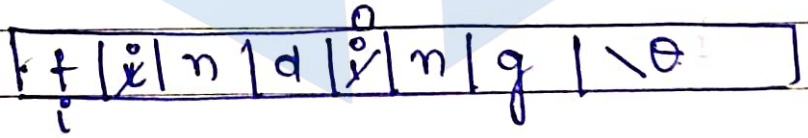
Findings Duplicates in a String



Method :-

1) Compare with other letters
2) using ~~Hastable~~ Hashtable or counting
3) Using Bit

1) Compare with other letters



Comparing A[i] == A[i+1], in whole string

```
int main()
{
    char str[10] = "Hello";
    for(int i=0; str[i] != '\0'; i++)
    {
        if ( str[i] != -1 )
        {
```

```
int count = 1;
for(int j=i+1; str[j] != '\0'; j++)
{
        if (str[i] == str[j])
        {
                str[j] = -1;
                count++;

        }

}
if(count>1)
{
    cout << str[i] << " is duplicated" <<
                count-1 << " times" << endl;

    }

}

}
Cout << "\n\n\n\n";

}
```

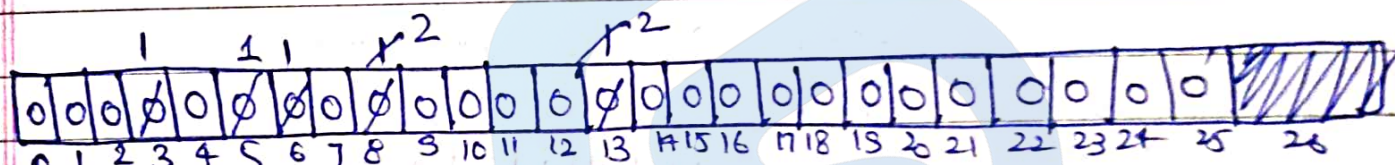
	102	105	110	100	105	110	103	
A	f	i	n	d	i	n	g	\0
	0	1	2	3	4	5	6	7

lowercase → 97-122

~~77~~ 97 - 122

a 8

o 25



f = 102 - 97

= 5

i = 105 - 97

= 8

n = 110 - 97

= 13

d = 100 - 97

= 3

g = 103 - 97

= 6

```
int main()
{
    char A[] = "finding";
    int H[26], i;

    for (i=0; A[i] != '\0'; i++)
    {
        H[A[i] - 97] += 1;
    }
    for(i=0; i<26; i++)
    {
        if(H[i] > 1)
        {
            printf("%c", i+97);
            printf("%d", H[i]);
        }
    }
}
```

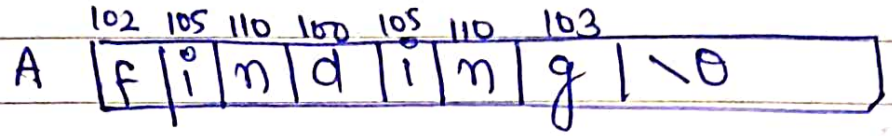
Time complexity $\rightarrow n + n$

$\rightarrow 2n$

$= O(n)$

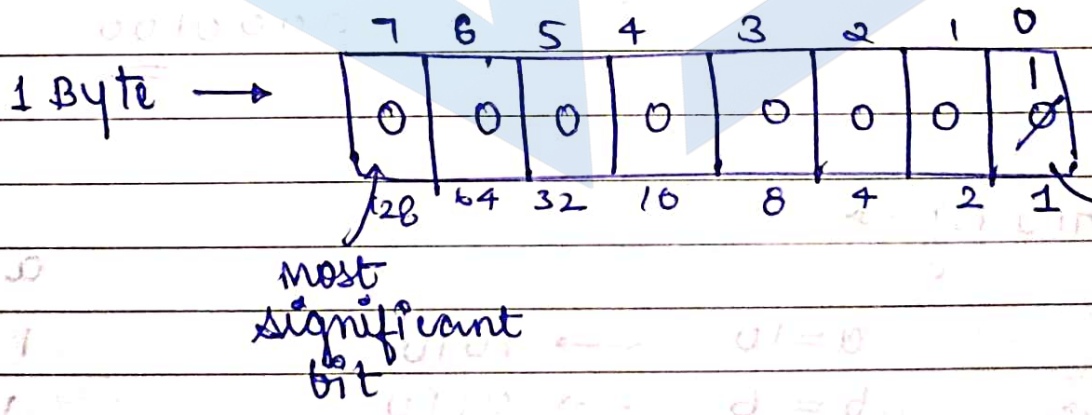
Pg. No.
Date / /

finding Duplicates in a String using Bitwise operator



Bitwise operations

- Left shift
- Bits ORing (Merging)
- Bit ANDing (Masking)



~~char H=0~~

↓

~~H=1~~

H=2 00000010

H=4 00000100

H = 8 → 00001000

H = 20 → 00010100

Now,

H = 1 → 00000001

H = H << 1

→ 00000010

↓ left shifting

H = 1 → 00000001

H = H << 2

→ 00000100

Anding →

a = 10 → 1010

b = 6 → 0110

a & b → 0010 ⇒ 2

a	b	&
1	1	⇒ 1
1	0	⇒ 0
0	1	⇒ 0
0	0	⇒ 0

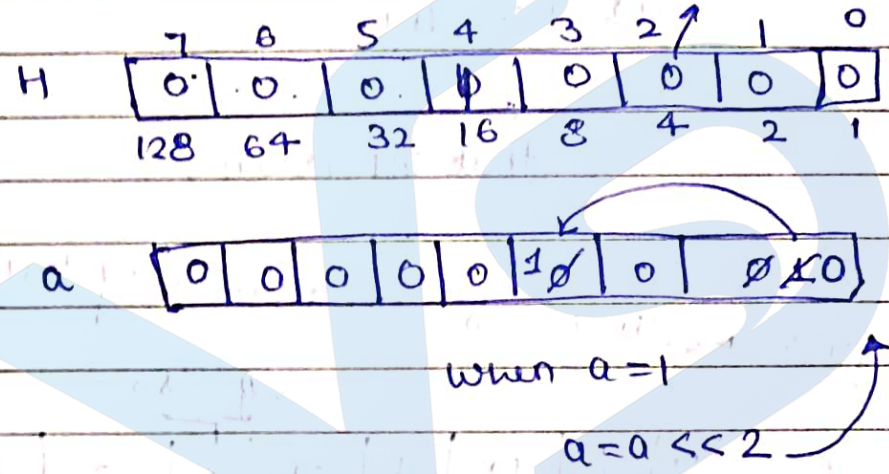
ORing →

a = 10 → 1010
b = 6 → 0110
1110 → 14

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

Masking → checking whether a bit is on or not is called masking

checking it is on or not



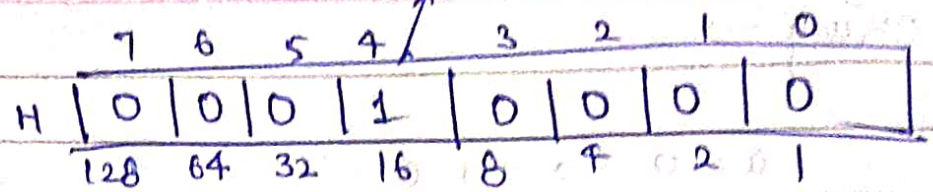
Anding → A H & a

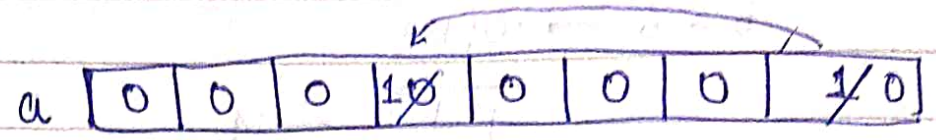
H & a = 00000000

If we got 0 only → then bit is off

If we got non zero → then bit is on.

checking this, it is on or not





a=1

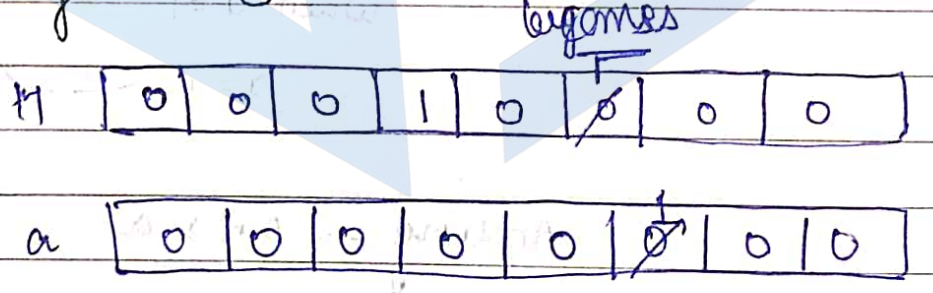
a=0<<4

H & a = true

↓

that bit is on.

Merging → Setting a bit on in memory known as merging



a=1

a=a<<2

H=a|H

↳

000|0|00

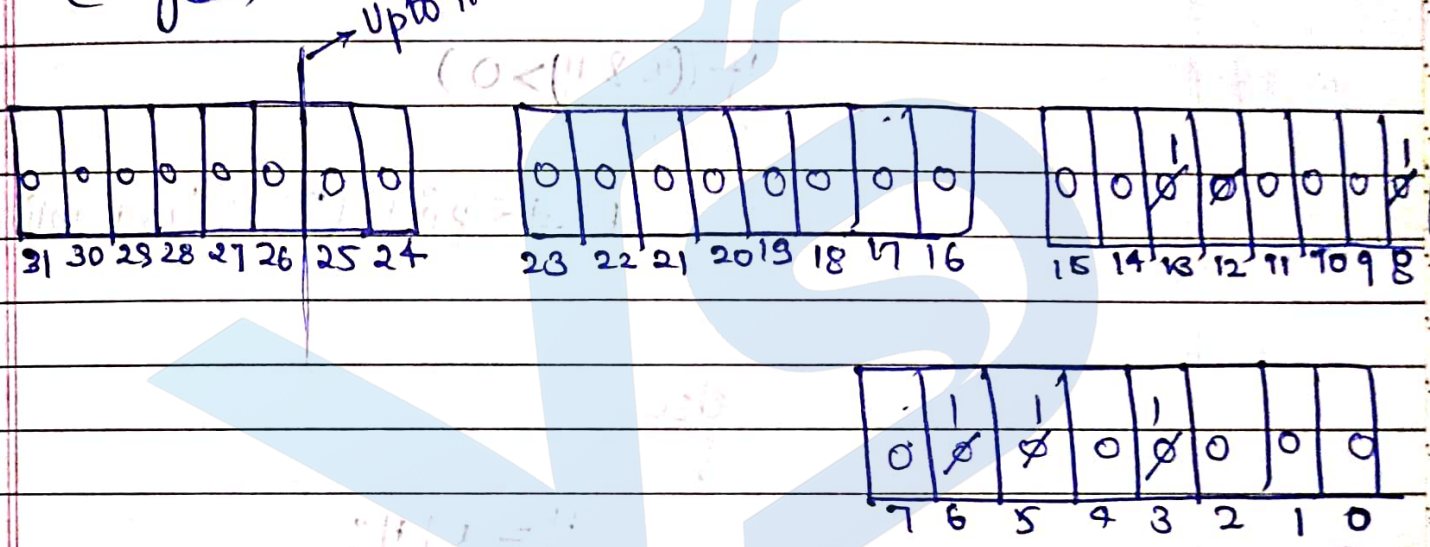
A | f | i | n | d | i | n | g | \0 |
0 1 2 3 4 5 6 7

26 - bits

↓

4 bytes ⟶ 32 bit

(int, long int)



long int H=0 ;

f = 102-97 = 5 ⟶ set bit on

i = 105 - 97 = 8 ⟶ set bit on

n = 110 - 97 → 13 ⟶ set bit on

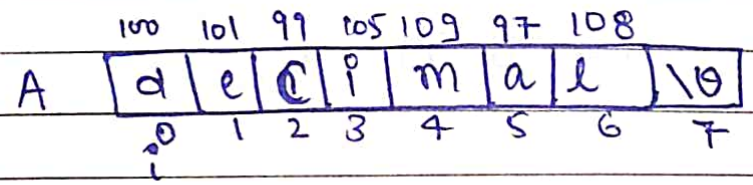
d → 100 - 97 = 3 ⟶ set bit on

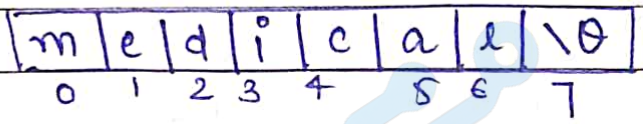
i → 105 - 97 = 8 ⟶ Already one, it means i is duplicate

```
int main()
{
    char A[] = "finding";
    long int H = 0, x = 0;

    for(i=0; A[i] != '\0'; i++)
    {
        x = 1;
        x = x << (A[i] - 97);

        if((x & H) > 0)
        {
            printf("%c is Duplicate", A[i]);
        }
        else
        {
            H = x | H;
        }
    }
}
```

Notebook
Pg. No.
Date / /

Checking if 2 strings are Anagram (distinct letters)





Two strings are said to be anagrams if they make a meaningful word by rearranging or shuffling the letters of the string. In other words, we can say that two strings are anagrams if they contain the same characters but in a different order.

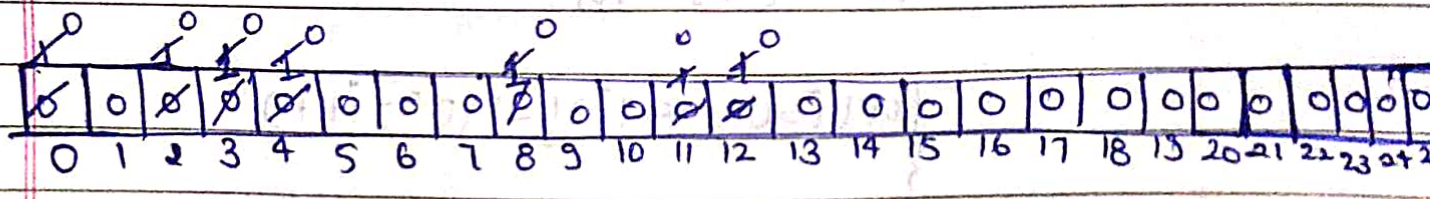
↳ size of two string

↓

equal size

Comparing two array in each letter

↳ $O(n^2)$



for A

d = 100 − 97 ⇒ 3 e = 99 − 97 = 2

e = 101 − 97 → 4 i = 105 − 97 ⇒ 8

m = 109 - 97 = 12

a = 97 - 97 = 0

l = 108 - 97 = 11

Time for Hash table

= n + n + n

= 3n

O(n)

for B

m = 109 - 97 = 12

e = 101 - 97 = 4

d = 100 - 97 = 3

i = 105 - 97 = 8

e = 99 - 97 = 2

a = 97 - 97 = 0

l = 108 - 97 = 11

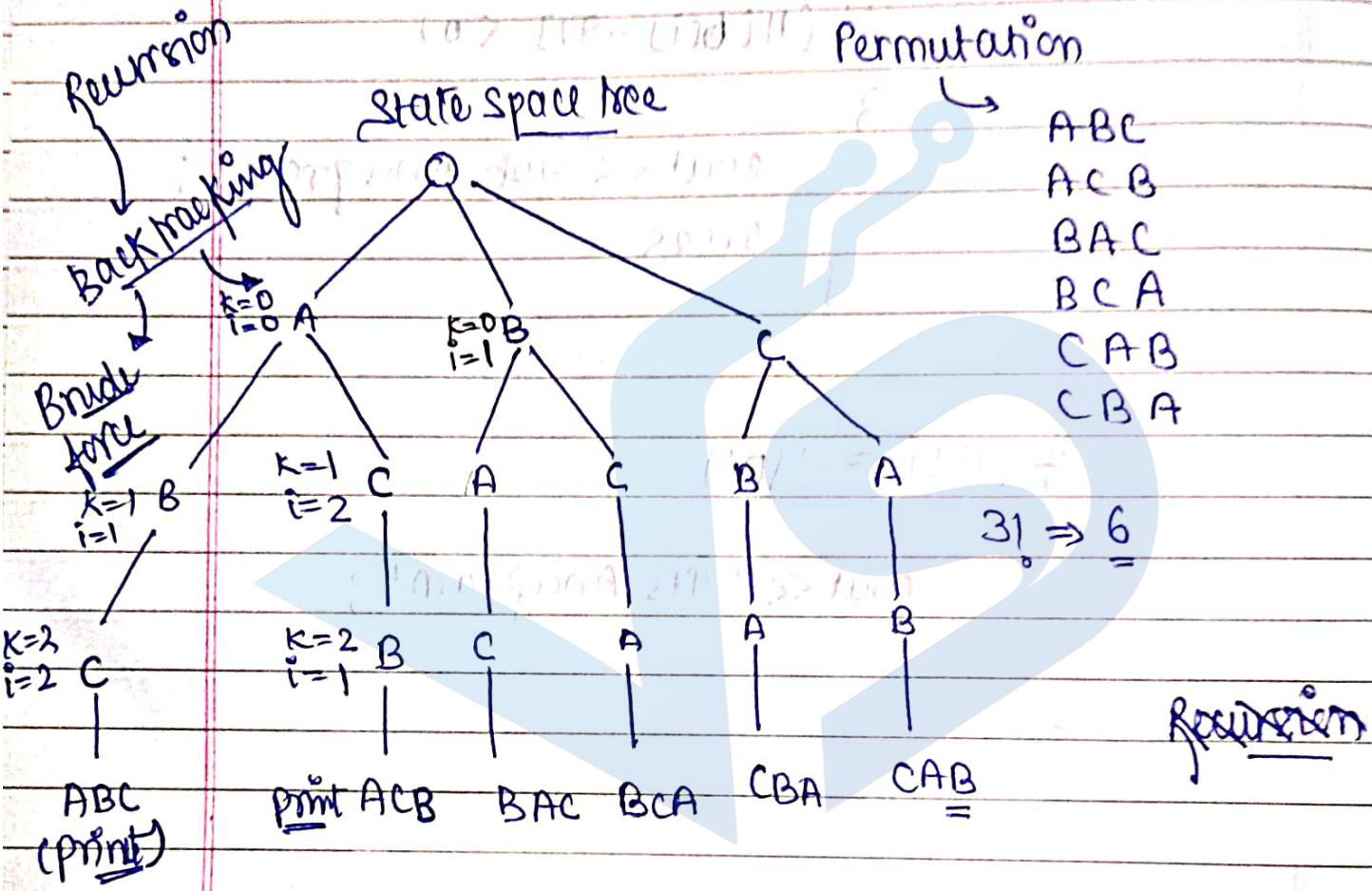
decrement when found match

```
int main()
{
    char A[] = "decimal";
    char B[] = "medical";
    int i;
    int H[26] = {0};

    for(i=0; A[i]!='\0'; i++)
    {
        H[A[i]-97]++;
    }
```

```
for(i=0; B[i]!='\0'; i++)
{
        H[B[i]-97]--;
        if(H[B[i]-97]<0)
        {
            cout<<"Not Anagram";
            break
        }
}
if(B[i]=='\0')
{
    cout<<"Its Anagram";
}
```

Permutation of String

S	A	B	C	\0
	0	1	2	3



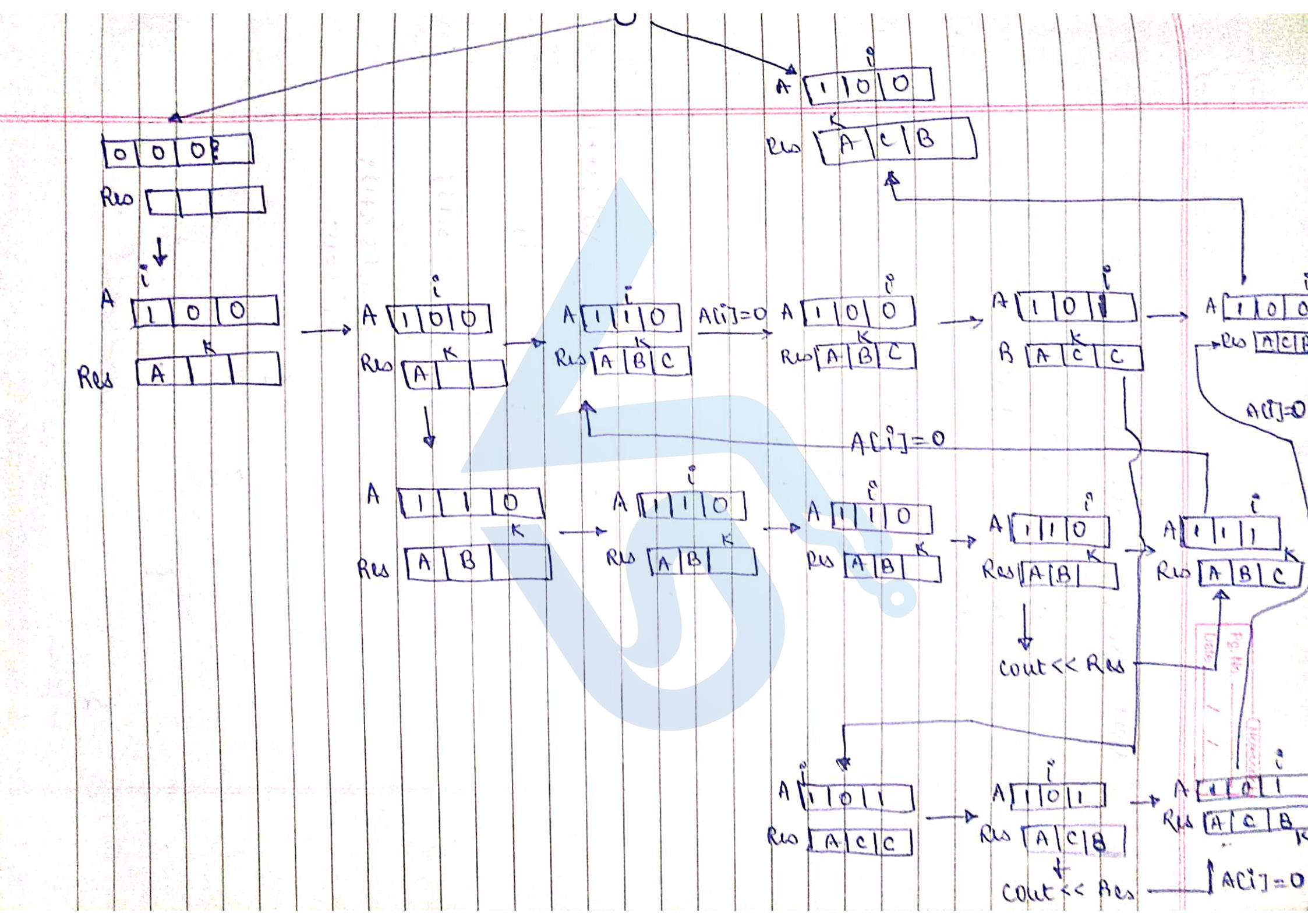
A Permutation of a string is another string that contains same characters, only the order of characters can be different.

A	1	1	0	0
	0	1	2	3

Res

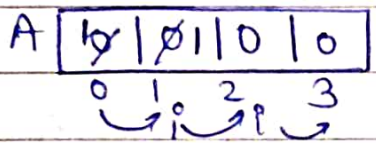
A	B	C	
0	1	2	3

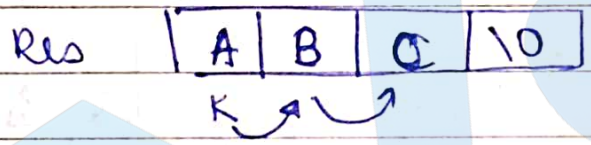
K → K


A[i]=0
A[i]=0
cout << Res
A[i]=0

```
void perm(char S[], int K)
{
    static int A[10] = {0};
    static char Res[10];
    int i;

    if (S[K] == '\0')
    {
        Res[K] = '\0';
        cout << Res;
    }
    else
    {
        for(i=0; S[i] != '\0'; i++)
        {
            if(A[i] == 0)
            {
                Res[K] = S[i];
                A[i] = 1;
                perm(S, K+1);
                A[i] = 0;
            }
        }
    }
}
```

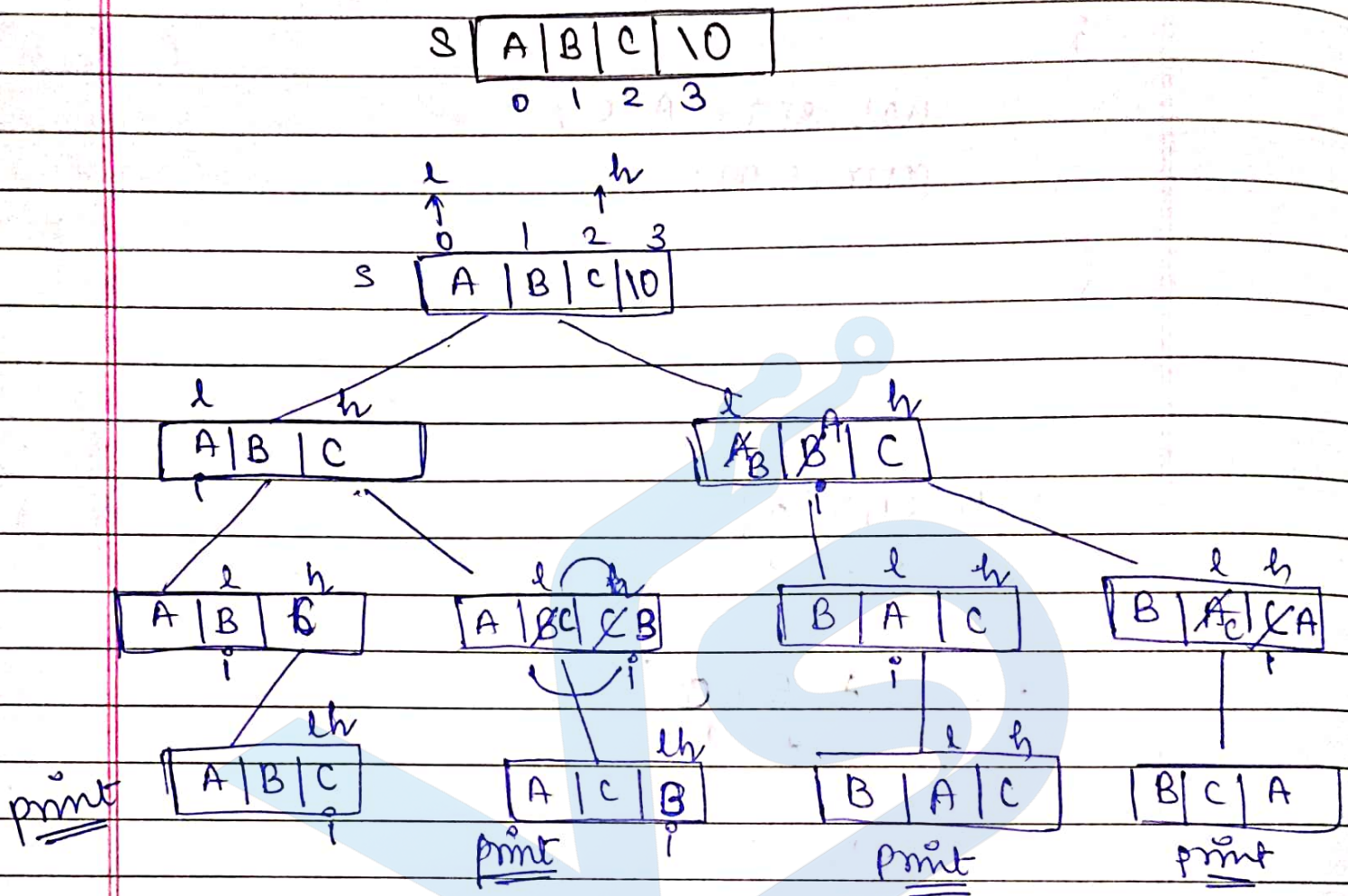
```
int main()
{
    char S[] = "ABC";
    perm(S, 0);
}
```





S[K] = \0

Method 2 :-




```
void Perm(char s[], int l, int h)
{
    int i;
    if(l==h)
    {
        cout<<s;
    }
    else
    {
        for(i=l; i<=h; i++)
        {
```

```
        swap(s[l], s[i]);
        perm(s, l+1, h);
        swap(s[l], s[i]);

    }

}

}
```