STRINGS

1. Character Sets / ASCII codes
2. Character Array
3. String
4. Creating a string

→ For English language

1. Character Sets / ASCII codes

American Standard Code for Information Interchange

A – 65	a – 97	0 – 48
B – 66	b – 98	1 – 49
⋮	⋮	⋮
Z – 90	z – 122	9 – 57

UNICODES

2 byte

16 bits

4x4 bits

Represented in form of hexadecimal

↲ enter – 10

Space – 13

esc – 27

$\boxed{0 - 127}$ Total 128

$2^7 = 128$

7 bits

1 byte

2. Character Array

```
char temp;
temp = 'A';
```

65 (Stored as 65)

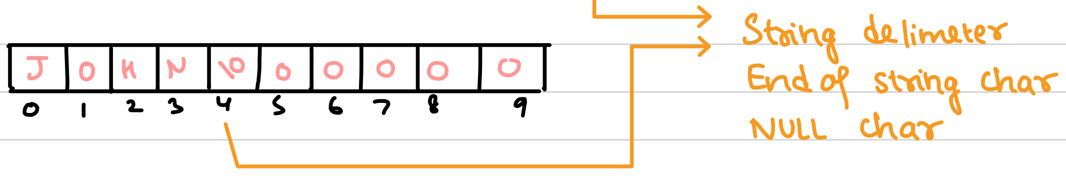
$\boxed{A}$

```
printf("%c", temp);
```

Displays 'A' and not 65

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

char x[5] = {65, 66, 67, 68, 69};
char name[10] = { 'J', 'o', 'H', 'N', '\0' };
```



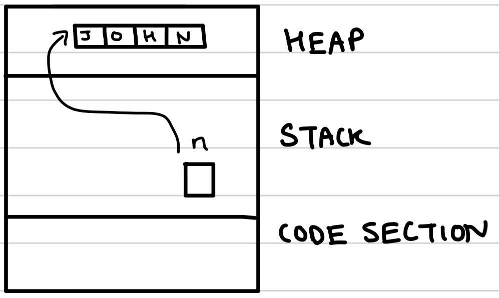
CREATING A STRING

```
(1)  char name[10] = { 'J', 'o', 'H', 'N', '\0' };

(2)  char name[] = { 'J', 'o', 'H', 'N', '\0' };      Size of array = 5

(3)  char name[] = "John";        compiler takes null character on its own

(4)  char *n = "John";
```



DISPLAYING A STRING

```
char name[10] = "David";
printf("%s", name);
scanf("%s", name);  →  cannot read after space
gets (name)  →  can read until you hit enter
```

FINDING LENGTH OF STRING

```
int main()
{
        char *s = "welcome";   // No size is required
        int i;
        for (i=0; s[i]'=0; i++)
        {
                     // This will remain empty
        }
}       printf("Length is %d", i);
```

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

i=7

CHANGING CASE OF A STRING

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

TOGGLE CASES

S	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]>= 'a' && A[i] <= 122)
                            A[i] -= 32;
        }
}
```

COUNTING NO OF VOWELS AND CONSONANTS

```
for (i=0; A[i] != '\0'; i++)
{
        if ( A[i) == 'a' || A[i) == 'e'..... )
                    vcount ++;

        else if (( A[i] >= 65 && A[i] <=90) || (A[i]>=97 && A[i]<=122))
                    ccount ++;
}
```

COUNTING NO OF WORDS

```
int main()
{
    char A[] = " How are     you ";
    int i, word =1;                 white space
                                    ↳ when there are more than one
                                                              space
    for (i=0 ; A[i]!='\0' ; i++)
         if(A[i]==' ' && A[i-1]!=' ')
              word ++;
}
```

VALIDATING A STRING

```
int valid ( char * name)
{
    int i;
    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))
              return 0;
        else
              return 1;
    }
}
```

```
int main()                 → This type of string is not modifiable
{
    char * name = " Anil321";
    if ( validate(name))
          printf(" Valid String");
    else
          Printf(" Invalid String");
}
```

FINDING DUPLICATES IN A STRING

```
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);
}
```

FINDING DUPLICATES IN A STRING USING BITWISE OPERATIONS

Manipulating bits of a byte

1. Left Shift <<
2. Bits ORing (Merging)
3. Bits ANDing (Masking)

most significant bit

7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0
128	64	32	16	8	4	2	1

least significant bit

(Represents how 8 is stored)

1 BYTE = 8 BITS

`char H=8;`

(1) LEFT SHIFT

`H = H << 2`

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0
128	64	32	16	8	4	2	1

(3) BITS ANDing

a = 10 → 1010

b = 6 → 0110

0010 → 2

(2) BITS ORing

a = 10 → 1010

b = 6 → 0110

1110 → 14

MASKING

H

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0
128	64	32	16	8	4	2	1

Q We want to know whether 4th bit in H is on or off.

a

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

Sol

a = 1

a = a<<4

a & H

a

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1
128	64	32	16	8	4	2	1

(perform ANDing)

MERGING

H

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0
128	64	32	16	8	4	2	1

a

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0
128	64	32	16	8	4	2	1

Q we want to set 3rd bit in H as on.

Sol

a = 1

a = a<<2

H = a | H

FINDING DUPLICATES

ASCII Codes

A

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

H

0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24
0	0	0	0	0	0	0	0
23	22	21	20	19	18	17	16
0	0	1	0	0	0	0	1
15	14	13	12	11	10	9	8
0	1	1	0	1	0	0	0
7	6	5	4	3	2	1	0

32 bits created for 26 alphabets

X

0	0	0	0	0	0	0	1

X also consists of 32 bits

```
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;
      }
}
```

** CHECK FOR ANAGRAM

When two strings are made up of same alphabets

A

100	101	99	105	109	97	108	
d	e	c	i	m	a	l	\0
0	1	2	3	4	5	6	7

B

m	e	d	i	c	a	l	\0
0	1	2	3	4	5	6	7

H

1		1	1	1				1			1	1													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

H

0		0	0	0				0			0	0													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

First check if the strings are of same length.

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

      for (i=0; A[i]!='\0'; i++)              // Making it 1
                H[A[i]-97]+=1;

      for (i=0; B[i]!='\0'; i++)              // Making it 0
                if (H[A[i]-97]<0)
                {
                          printf(" Not Anagram");
                          break;
                }

      if (B[i]=='\0')
                printf(" Anagram");
```

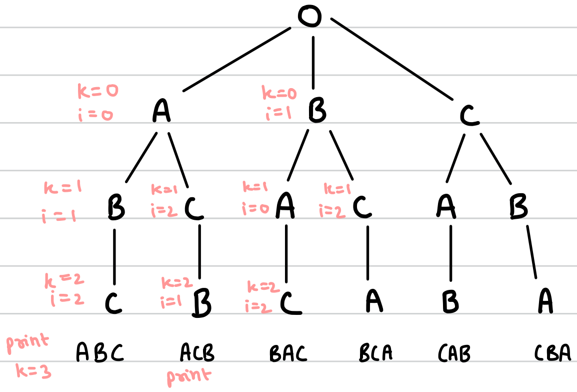
PERMUTATION OF A STRING

1st Method

A	B	C	\0
0	1	2	3

3!
n!

State Space Tree



Brute Force: Finding all possible permutations

Back Tracking

- Implemented using recursion
- To achieve brute force.

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

    if (s[k]=='\0')
    {
        Res[k] = '\0'
        printf("%s", 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;
            }
        }
    }
}
```

S

A	B	C	\0
0	1	2	3

A

0	0	0	0
0	1	2	3

Res

A	B	C	\0
0	1	2	3

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

2nd Method

```
void perm(char s[], int l, int h)
{
    int i;
    if (l==h)
        printf("%s", S);
    else
    {
        for(i=l; i<=h; i++)
        {
            swap(s[l], s[i]);
            perm(s, l+1, h);
            swap(s[l], s[i]);
        }
    }
}
```