1000	1004	1008	1012	1016
10	20	30	40	50
a[0]	a[1]	a[2]	a[3]	a[4]

→ eg:- {

```
void main()
int a[5] = {10, 20, 30, 40, 50};
int ele;
ele = sizeof(a)/sizeof(a[0]); //
pf("No. of elements in a = %d", ele);
}
```

// Total ele size / size of one element

→ To print elements in array:-

```
for(i=0; i<ele; i++)
pf("%d", a[i]);
pf("\n");
```

o/p:- 10 20 30 40 50

→ To print the elements in Reverse:-

```
for(i=ele-1; i>0; i--)
  pf("%d", a[i]);
  pf("\n");
```

Consider;

int a[5] = {10, 20, 30, 40, 50};

Here the no. of elements are more than the size of an array

But the compiler will consider the size on the basis of int n[2] so it will allocate only n no. of elements.

Note:- If we initilize an array partially, compiler will put remaining elements as zeros.

→ eg:- int a[5] = {10, 20, 30} // Partially initialised array

o/p:- 10 20 30 0 0 ← Remaining elements are zero

If no. of elements are not given, the compilers will look for initialization.

→ if a[ ] = {10, 20, 30};

→ when you are declaring an array providing the no. of element is mendatory.

→ when you are initializing an array, providing no. of element is optional

Scanning the array elements and printing on the screen:-

Eg:- void main()
{
int a[5], ele, i;
ele = sizeof(a)/sizeof(a[0]); // → More generalise
→ // to scan the element
printf("Enter the elements \n");
for (i=0; i<ele; i++)
scanf("%d", &a[i]);

→ // to print the elements
for (i=0; i<ele; i++)
Pf("%d", a[i]);
Pf("\n");

→ // To print the elements in reverse
for(i=ele-1; i>=0; i--)
Pf("%d", a[i]);
Pf("\n");
}

Dynamic declaration:-

Write a program to scan an array of 10 integer, after scanning, print it, after printing reverse the element of the array, again print it.

It is not reverse printing, the element have to be reversed.

0-9
1-8
2-7
3-6
4-5

```
#include<stdio.h>
void main()
{
int a[10], ele, i, j, t;
ele = sizeof(a)/sizeof(a[0]);
printf("Enter the element\n");
for(i=0; i<ele; i++)
  scanf("%d", &a[i]);
printf("Before reverse\n");
for(i=0; i<ele; i++)
printf("%d", a[i]);
printf("\n");

for(i=0, j=ele-1; i<j; i++, j--)
  {
    t=a[i];
    a[i]= a[j];
    a[j] = t;
  }
printf("After reverse\n");
for(i=0; i<ele, i++)
  printf("%d", a[i]);
  printf("\n");
    }
```

Session - 39

31-7-20

→ Secondary datatypes (Array)

(Memory Related Discussion)

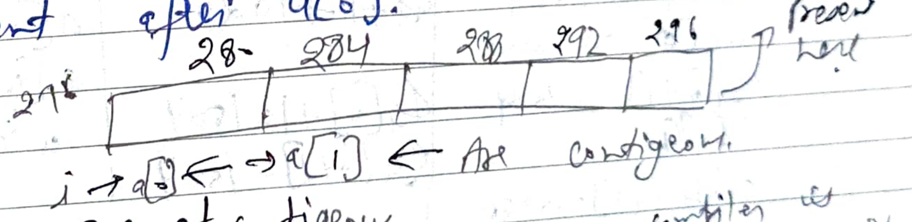
```
{
int a[5] = {10, 20, 30, 40, 50};
for(i=0; i<5; i++)
pf("%u", &a[i]);
pf("&i = %u\n", &i); // %p can be used
}                    // %u gives warning
```

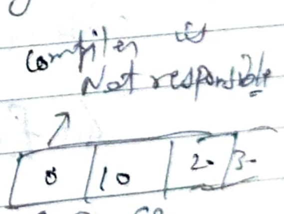
Here can't discuss properly %u

o/p :- 38196 5280
" " 84
" " 88
" " 92
" " 96
&i = " " 276

→ 'i' may be before 'a' or after 'a' location.

→ a[0] = a[1] are contigeous (within array) means compile gives gurranty a[1] is present after a[0].





→ a[-1] = 200;
pf("i = %d\n", i);
o/p :- i = 200

because i got disturbed

→ If we are using beyond the limit compiler is not responsible, programmer is responsible.

→ If we write int a[5];
int b[5];

we can't say these 2 are contigeous

{

int a[5] = {10, 20, 30, 40, 50};

Array name gives base/starting address

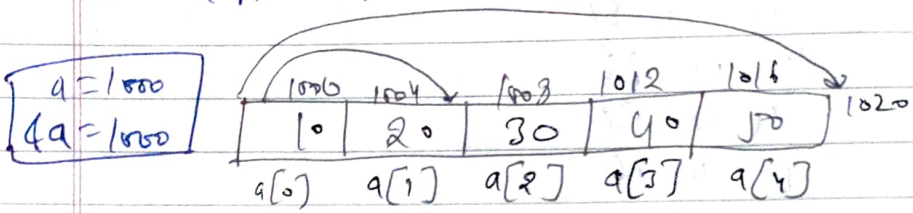
printf("a = %p &a = %p\n", a, &a);

printf("a+1 = %u &a+1 = %u\n", a+1, &a+1);

means next element address

o/p:- a = 3850 6104 b = 3850 6104

a+1 = 3850 6108 &a+1 = 3850 6124



→ &a+1 → Total array size +1

→ a+1 → One element gets incremented.

a → 1000	&a → 1000	a[0] → 10
a+1 → 1004	&a+1 → 1020	a[0]+1 → 11
&a[0] = 1000		a[0+1] → 20
&a[0+1] = 1004	&a[0]+1 = 1004	a[1]
&a[1]	&a[0+1] = 1004	
a[0]+a[1] = 10+20 = 30		

&a[0] + &b[0] = ✗ we can't add address as like in case of pointer

Note: a[i] == i[a] == *(a+i)

Array can be represent in the form of pointers.

*(a+1) = 1004

a = 1000
&a = 1000
&a[0] = 1000

→ Array name represents base address of element.

→ Array name is treated as constant pointer.

int * const p;

#
```
{
int a[5]={10,20,30,40,50};
for(i=0;i<5;i++)
printf("%d %d %d\n",a[i],i[a],*(a+i));
}
```

o/p:-
10 10 10
20 20 20
30 30 30
! ! !

a[i] == i[a] == *(a+i)

&a[2] can be written as ~~&~~*(a+2)
(cancelled address & dereference) = (a+2)

#
```
{
int a[5],i;
for(i=0;i<5;i++)
printf("%d",i+a);
for(i=0;i<5;i++)
printf("%d %d %d\n",a[i],i[a],*(a+i));
}
```

o/p:-
0 1 2 3 4
10 10 10
20 20 20
30 30 30
! ! !

a++ → ✗ it is an error
a = a+1 → ✗ error
a+1 → No error

Array treated as const pointer
Can't modify.

3-8-20

Session - 40:

→ Secondary datatypes (Array)

int a[5] = {10,20,30,40,50};

1000	1004	1008	1012	1016
10	20	30	40	50
a[0]	a[1]	a[2]	- - -	

int *p;
p = a ✓
a = p ✗

a → 1000 a+1 → 1004
&a → 1000 &a+1 → 1020 ← Total array size
&a[0] → 1000 &a[0]+1 → 1004 gets incremented.

a++; X error

a[i] == *(a+i) == *i[a]

*a+1 = 1

*(a+0) +1
a[0] +1
10+1 = 11

*(a+1) = 20

**a → X error

This is because 'a' is of an array type and it is a 1-D array & it is being treated/tried to de-ref two times. So Compiler thrown error

&i+1 → integer address incremented by 4 byte.
Note:- Array name gives us the address, not the value.

# {
int i=10;
int a[5] = {10, 20, 30, 40, 50};
pf("%d\n", *a[i]); //error ? Time dereference (*)([])
}

eg:- { int i=10;
int a[5] = {10, 20, 30, 40, 50}, *p;
p=a; ← Base address of 'a' array
// p=&a; ← Warning because '&a' means that total array address not the starting address
for(i=0; i<5; i++)
pf("%d ", *p++); // p is post inc [assign then inc]
pf("\n");
o/p:- 10 20 30 40 50.

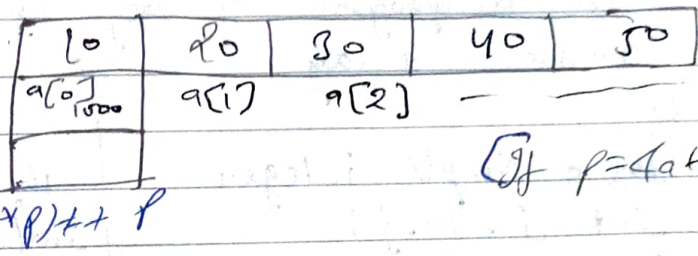
If the below statement is written instead of this,
pf("%d ", (*p)++); // value inc

Consider the entire program:-

```
for (i=10; i<15; i++)
pf ("%d", (*p)++);
pf ("\n");
for (i=0; i<5; i++)
pf ("%d ", a[i]);
pf ("\n");
}
```

o/p:-

10	11	12	13	14
15	20	30	40	50



[If p=a+3 = 60 bytes inc]

// p = a+3 → this ptr points to the element 40

```
pf("%d", p[-2]);
}
```

10	20	30	(40)	50

pointer points here

If a[-1] = 30

If a[-2] = 20

⇒ If p is taken as char *,

eg:- char *p;

p = a // Warning

```
for(i=0; i<5; i++)
pf("%d", *p++)
}
```

(postinc pointer)

o/p:-

10 0 0 0 20

⇒ (*a+1) is not equal to a[1] ∈ where 'a' is a[0]

a[0]+1

where,

*(a+1) → a[1]

classmate
Date 4-8-20
Page

Session – 4

→ Secondary data types (array)

*a + 1 — a is an array type, but the exp is pointer type. So convert the exp into array type

⇒ a[i] == *(a+i)

⇒ *a + 1

⇒ *(a+0) + 1

Write a program to copy one array elements to another array.

→ eg:

for a simple integer	But for array
int i = 10, j;	int a[5] = {10,20,30,40,50};
to copy the content of i to j	int b[5];
j = i;	b = a; ✗ error because compiler considers array as a constant pointer
j = 10	

The element have to be copy one by one.

⇒
```
{
int a[5] = {10,20,30,40,50};
int b[5];
ele = sizeof(a)/sizeof a[0]; // To find the no of ele in array
// copy the element from a to b
// b=a; ✗ error
for (i=0; i<ele; i++)
b[i] = a[i];
// printing elements of array 'a'
printf("\n --- a --- \n");
for (i=0; i<ele; i++)
printf("%d", a[i]);
// printing elements of array 'b'
printf("\n --- b --- \n");
```

```
for(i=0;i<ele;i++)
  pf("%d",b[i]);
  }
```

o/p:- ---- a ----
10 20 30 40 50
------ b ------
10 20 30 40 50

⇒ To copy the array elements in reverse order.
main logic
for(i=0;j=ele-1;i<ele;i++,j--){
b[i]=a[i];}

Find out how many prime nos are there in an array.
* main logic:-

```
for(i=0;c=0;i<ele;i++)
{
  for(j=2;j<a[i];j++) // prime no. logic
  {
    if(a[i]%j==0)
      break; // comes out of inner loop
    if(j==a[i])
    {
      c++;
      pf("%d",j);
    }
  }
  pf("\n");
}
```

Session — 42 5-8-20

→ Secondary data types (Array)

Finding the second largest element in an array

```
{
int a[10] {1,9};
int i,ele, L, SL;
ele = sizeof(a)/sizeof(a[0]);
pf("Enter the ele \n");
for(i=0; i<ele; i++)
sf("%d", &a[i]);
pf("\n");
if(a[0] > a[1])
  {
    L = a[0];        ①
    SL = a[1];       ②
  }
else
  {
    L = a[1];        ①
    SL = a[0];       ②
  }
for(i=2; i<ele; i++)
  {
  if(a[i] > L)
    {
      SL = L;        ②
      L = a[i];      ①
    }
else if (a[i] > SL)   // && a[i] != L
    SL = a[i];
  }
pf("SL = %d \n", SL);
}
```

Bug removed

Session → 49

Bubble sort :-

eg: given array :- | 25 | 15 | 17 | 10 | 5 |

expected op :- | 5 | 10 | 15 | 17 | 25 |

1st iteration

{25, 15, 17, 10, 5}
These two are compared
{15, 25, 17, 10, 5}
These two compared
{15, 17, 25, 10, 5}
{15, 17, 10, 25, 5}
{15, 17, 10, 5, 25}

Largest element moved at the end (sorted)

2nd iteration

{~~15, 17, 25, 10~~,
{15, 17, 10, 5, 25}
{15, 17, 10, 5, 25}
{15, 10, 17, 5, 25}
{15, 10, 5, 17, 25}
{15, 10, 5, 17, 25}

2nd largest element

So on - - - -

outer loop variable ele-1 times

i=0	i=1	i=2	i=3
0-1	0-1	0-1	0-1
1-2	1-2	1-2	
2-3	2-3		
3-4			

0	1	2	3	4
25	15	17	10	5

1st 2nd

1st index & 2nd index are compared
if 1st index ele > 2nd index ele, then swap the element.

Inner loop

i=0	Swaps
0	4
1	3
2	2
3	1

```
{ //main logic
  for(i=0; i<ele-1; i++)
  {
    for(j=0; j<ele-1-i; j++)
    {
      if(a[j] > a[j+1])
```

// Loop starts with zero bec every time 1st index ele is taken into consideration.

```
{
  t = a[j];
  a[j] = a[j+1]
  a[j+1] = t;
}
}}
```

// To short in decending put if(a[j] < a[j+1])

// j < ele - i - 1 ele = 5
when i = 0, j = 0
0 < ele → 0 < 4

Selection sort:

Ascending order in selection sort

In selection sort, a particular index & that index value is compared with all the values and sorted.

1st iteration

{25, 15, 14, 10, 5
swap & (1st > 2nd)
{15, 25, 14, 10, 5}
No swap
{15, 25, 17, 10, 5}
{15, 25, 17, 10, 5}

2nd iteration

{15, 25, 17, 10, 5}
{15

```
for(i=0; i<ele-1; i++)
{ for(j=i+1; j<ele; j++)
  { if(a[i] > a[j])
    { temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }}}
```

Char array vs String

```
{
char a[] = {'a', 'b', 'c', 'd'};
char s[] = "ABCD"; // 'ABCD' is string constant
pf("%ld %ld\n", sizeof(a), sizeof(c));
}
```

→ Char array → Collection of characters
→ String → Collection of characters ended with '\0' (slash zero)

① \0 indicates end of valid characters.
② \0 is one of the escape sequence.
③ \0 requires 1 byte memory to store

④ \0 means in one byte, all bits are zeros '0'
⑤ '\0' is not a printable character.

How to scan a string and how to print it?

```
{
char s[10];
pf("Enter the string\n");
sf("%s",s);        // %s should be given for(s)
pf("s= %s\n",s);
}
```

// When enter button is pressed, scanf stop scaning the data & it automatically puts '\0' in the next location '\0' acts as seperator blw valid i/p & junks

i/p:-
Hello ↵
Hello

Note:- In string case, the address (%s) has to be given in both printf() and scanf().

```
pf("s+1 = %s\n",s+1);
```
the size is incremented by one byte bec. char array

`s=s+1;` ✗ error because char array treated as constant pointer

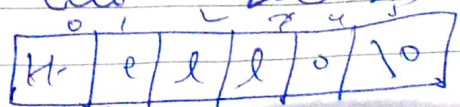
⇒ whenever '%s' is used, after space, the input is not considered.
eg:- Enter the string
abc_de
s= abc

✗ To scan the space

```
sf(" %[^\n]",s);   // Array name represents the base address so no need of '&' symbol while scanning
pf("s= %s\n",s);
```

→ 's+1' prints 'ello' because,

's+1' prints from



Session → 44

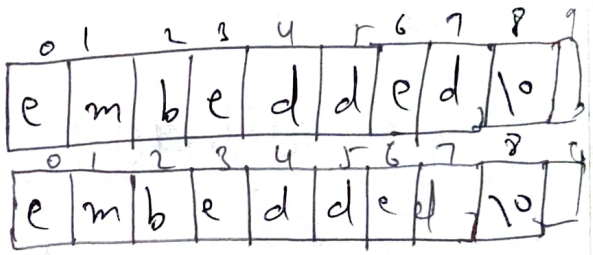
8-8-20

Write a program to copy a string from source to destination.

```
#include<stdio.h>
Void main()
{
  char s[10],d[10];
  Printf("Enter the string,s\n");
  Scanf("%s",s);
  for(i=0; s[i] ;i++)
   d[i] = s[i]; // d[i] = '\0';→ because, the condition
   d[i]='\0';          fails at [a] due to '\0'
   Printf("s= %s  d= %s\n",s,d);
}
```

O/P:-

Enter the string,s
embedded
s= embedded d= embedded



Write a program for reversing the given string

```
#include<stdio.h>
  void main()
{
    char s[10],t;
    int  i, j ;
    Printf("Enter string..\n");
    scanf("%s",s);
    Printf("Before reverse s= %s\n",s);
    for(i=0;s[i];i++); // finding the length of
                          string
```

```
for(j=0, i=i-1; s[i]; i--, j++) // reversing
    {
        t= s[i];
        s[i] = s[j];
        s[j] = t;
    }
    printf("After reverse s= %s \n", s);
}
```

O/P :-

```
Enter string
embedded
before reverse
s= embedded
After reverse s= deddebme
```

Write a program to compare two strings

```
void main()
{
  char s[10], s1[10]
   int i;
  printf("Enter s and s1 \n");
   sf("%s %s", s, s1);
  for(i=0; s[i]; s1 i++)
    {
      if (s[i] != s1[i])
        break;
    }
  if(s[i]==s1[i])
   printf("Equal \n");
   else
  printf("Not equal \n");
}
```

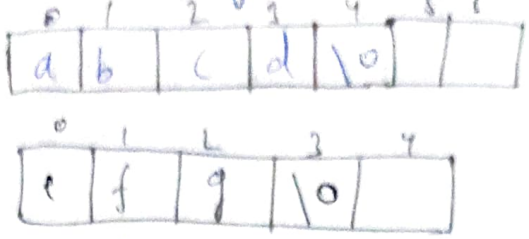
O/P :-

```
Enter s and s1
abcd
ABCD
Not equal..
```

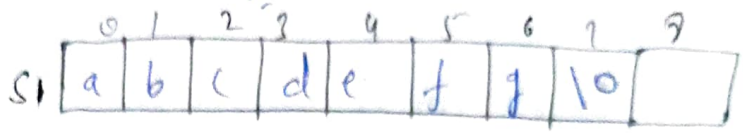
String concatenation

Means 2nd string should be added at the end of the 1st string

eg:-



After concontention




```
{
 char f[20], s[10];
 Pf("Enter 1st string\n");
 Sf("%s",f);
 Pf("Enter 2nd string\n");
 Sf("%s",s);
 Pf("Before f=%s  s=%s\n",f,s);
 for(i=0;f[i];i++); // Dummy loop for finding the length of f[i]
 for(j=0;s[j];j++,i++)
   f[i] =s[j];
   f[i] = s[i];// for putting \0
```

Session – 45 — 10 Aug 2020

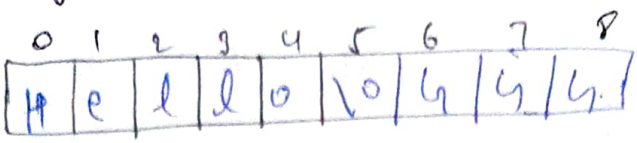
using gets(); → only strings can be scanned

puts(); → only a string can be printed

eg:-

```
{ char s[20];
  Pf("Enter the string\n");
  gets(s);
  puts(s);
}
```

Write a program to print the string char by char



Example program

Don't use gets() for scanf()
puts() for printing printf() together
due to some buffer related issues
the compiler may give
unexpected output

```
#include <stdio.h>
void main()
{
    char s[10];
    int i;
    printf("Enter the string\n");
    scanf("%s",s);
    printf("s=%s\n",s)//print the input as it is

//for(i=0;i<10;i++)//→ this prints even garbage value
//for(i=0;s[i]!=0;i++)//this will prints the value until
for(i=0;s[i];i++)//↗                    it finds \0
    printf("%c",s[i]);// %c is used as we printing
    printf("\n");                    the character
}
```

o/p

```
Enter the string
Hello
s=Hello
H e l l o
```

write a program to find the length of the string.

```
{
    char s[10];
    int i;
    printf("Enter the string\n");
```

```
scanf("%s", s);   // %s does not consider space
for(i=0; s[i]; i++);
printf("The length of %s = %d\n", s, i);
}
```

Write a program to convert a given string into upper case

each char has to be checked if it's in lower case or upper case

If lower case, convert it into upper case

If capital → don't change

```
{
char s[10];
int i;
pf("Enter the string\n");
sf("%s", s);
pf("Before = %s", s);
for(i=0; s[i]; i++)
{
if(s[i]>='a' && s[i]<='z')
s[i] = s[i]-32;
}
printf("After s= %s\n", s);
}
```

o/p:- Hello
HELLO.

Session = 46 11-8-20

Write a program to count how many times a given char is present in a given string.

```
#include <stdio.h>
Void main()
{
  char  s[10], ch;
  int   c, i;
  Printf("Enter the string\n");
  scanf("%s",s);
  Printf("Enter the char\n");
  scanf(" %c",&ch); // space is provided, so that the scanf() waits for the user to give the i/p else scanf() is not wait for the input.
  for(i=0;c=0;s[i];i++)
   if(s[i]==ch)
     c++;
  Printf("In %s %c is present %d times\n",s,ch,c);
}
```

o/p:-
Enter the string
embedded ↵ d
In embedded d is present 2 times

Replace one char by another.

eg:- embedded ↵
1st char → e
2nd char → a
after = ambaddad

```
#include<stdio.h>
 void main()
 {
   char s[10], ch, ch1;
   int i;
```

```
Pf("Enter the string\n");
Sf("%s",s);
printf("char1 Enter\n");
Sf("%c",&ch);
Pf("Enter char2 \n");
Sf("%c",&ch2);
Pf("Before replace %s\n",s);
for(i=0;s[i];i++)
    if(s[i]==ch)
    s[i]=ch1;
Pf("After %s\n",s);
}
```

To delete the given char in a string

i/p → e m b e d d ed
'e'

o/p → m b d d d

```
#include<stdio.h>
void main()
{
    char s[20],ch;
    int i,j;
    Pf("Enter the string\n");
    Sf("%s",s);
    Pf("Enter the char\n");
    Sf("%c",&ch);
    Pf("Before s=%s\n",s);
    for(i=0;s[i];i++)
    {
        if(s[i]==ch)
        {
            for(j=i;s[j];j++)
                s[j]=s[j+1];
        }
        i--; //
    }
```

//i-- is used to remove the bug which is repeating the char

// a a b c d → removing a
o/p: a b c d

if i-- used
b c d is o/p

To delete the first occurance

```
if(s[i]==ch)
{
    for(j=i; s[j]; j++)
      s[j]=s[j+1];
}  break;
```

Difference b/w char array and char pointer

char array initialization	char pointer initialization
→ char s[] = "Vector India";	→ char *p = "Vector India"
sizeof(s) = 13; bytes	sizeof(p) = 8 bytes
V e c t o r I n d i a \0	→ V e c t o r I n d i a \0
→ String constant represents the base address	p — pointer pointing address

eg:-

```
void main()
{
    char s[]="ABCD";
    char *p="abcd";
    printf("%ld %ld\n", sizeof(s), sizeof(p));
}
```

→ A B C D \0

← 5 → 5 bytes

(memory is same for both s & string constant)

→ A B C D \0 → (5) bytes

p [300] (8 bytes)

Here (+1) / 1 byte involved

(as p is char pointer, memory is differ for pointer & string constant).

→ s & string constant both store in stack section

→ As s is array, it is constant pointer.

→ Array as pointer representation

s[0] = *(s+0)

→ s[6] = 'x'; is allowed

→ s="Bangalore"; is Not allowed

→ Total 13 bytes required to store

→ p is stored in stack section whereas string in code section

→ As p holding address of code section so it is a read only pointer

Pointer as array representation

*(p+7) = p[7]

→ p[6] = 'x'; is Not allowed

→ p="Bangalore"; is allowed

→ Total 13+8 = ~~17~~ 21 bytes required to store

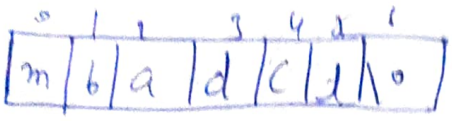
Session = 47

12-08-20

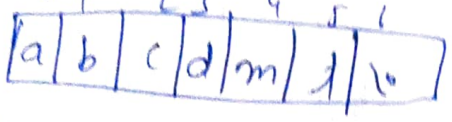
```
{
  char a[] = "ABCD";
  char *p = "abcd";
  //pf("%s %s\n", a, p);   // ABCD  abcd
  // pf("%c", a[0]); //A     pointer represent in array form
  //pf("%c\n", p[0]);  //a
  //pf("%c", *(a+0));  //A
  //pf("%c", *p);   // a
  // pf("%s", *p)  // Segmentation fault because p is address not *p
                   // → p will be used. o/p abcd
  //pf("%s\n", a[1]); //a[1] is a value not address so again error
                      %c will be used
  // p=a;  //error
  // a=p;  ✓
  // a = "mnop" // error  const pointer
  // p = "MNOP" // No  error
```

Bubble Sort (Sort String)

before



After



Bubble sort

```
→ char s[20], k;
  int i, j, l;
  printf("Enter the string\n");
  scanf("%s", s);
  Pf("Before s = %s\n", s);
  for(l=0; s[l]; s++); //to print the length  find
  for (i=0; i<l-i; i++)
    {
      for(j=0; j<l-1-i; j++)
        {
          if(s[j]<s[j+1])
            {
              t= s[j];
              s[j] = s[j+1];
              s[j+1] = t;
            }
        }
    } printf("After s= %s\n", s);
}
```

O/P Enter the string
abndgc
before = s = abndgc
After = s = abdegn

Session — 48 13-8-20

Functions

→ Function is a block where code is written
→ Function is a set of instructions placed together to perform specific task.
Multiple blocks constitute a program.

Types of functions: Application Program Interface

1. Predefined / Library fun (Also called as API functions)
2. User defined function
→ Using this repetitation of code gets reduced
⇒ .exe file size gets reduced.

To create & use function you must know these three :-

1. Function declaration / Prototype
2. Function call
3. Function definition — For user defined function

1st method

declare above main

```
main()
{
  call in main;
}
define below main
```

2nd method

if a function is defined above main, there is no need to declare it

```
fun def.
main()
{ fn call;
}
```

For predefined function
declaration is present in header file.
definition is present in libraries