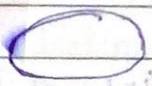
29-1-18

Basic 'C'

★ flowchart

Diagmodaly way of representing a Prg

 → start / stop

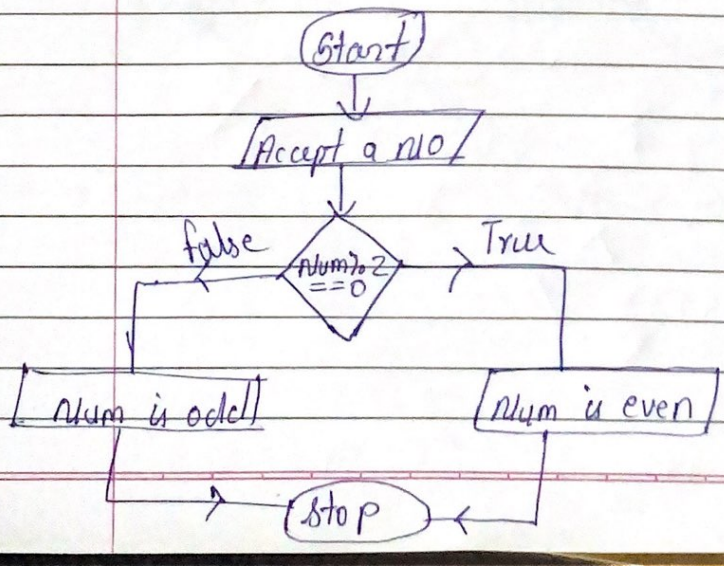
→ ↓ → flow line

 → process

 → i/p or o/p

 → Decision

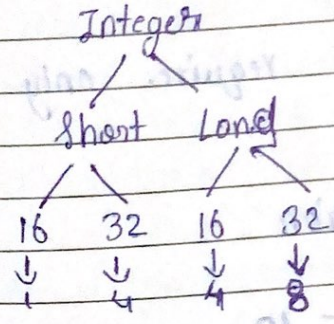
1 → 10/2 → 5 (quotient)
% → 10/2 → 0 (remainder)
== → comparision
if true rd 1 or else ret 0



Variable → A name given to memory location in which value can be stored is called as variable.

Rules → 1] It should not use keyword as variable
2] The variable cannot have wide space.
3] The format must be Alphanumeric, (a1, b1, c1)

format Specifie	Datatype	keywords	16 bit	32 bit
%d	Integer	int	2	4
%f	float	float	4	4
%lf	Double	double	8	8
%c	Character	char	1	1

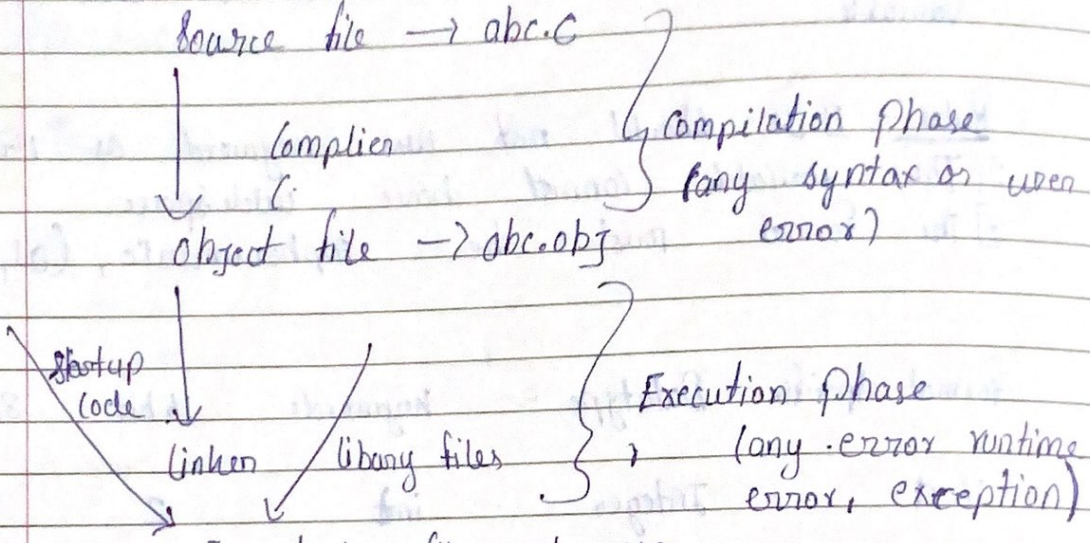


int a=10;
↓
Assingment operator

ASCII
a...z, A...Z, ↵, space
↓
0 to 255

BARFI Page No. Date

Conversion of '.c' to '.exe'



Operator :-

They are the special symbol which operate on operand

→ 3 types

i] Unary operator → It require only one operand

a] uncary minus (-)

use for negation

```
int a=-10;
a=-a;        o/p:- 10
```

b] increment operator → (++)

increment the value of variable by 1

```
int a=10;
a++; // a=a+1  a=+1        o/p:- 12
++a; // a++
```

Nothing on left side incr

It is divided in 2 types

1] Pre - increment :
2] Post - increment :

Pre - increment → ++a

```
int a = 10;
int b;
b = ++a;        a,b = 11,11
```

```
--b
a,b = 9,9
```

Post increment → a++

```
int a=10;
int b
b = a++;        a,b = 11,10
```

```
b--
a,b = 10,9
b=10
a=9
```

c] Decrement (--)

d] Size of ( ) = return size in bytes.

```
a=10, ch='x';
pf( %d sizeof(a); //4
  ( %d sizeof(ch); //1
  ( %d sizeof(10); //4
  ( %d sizeof(10.0); //8 (Double by default
  ( %d sizeof(10.0f); //4
  ( %d sizeof(double); //8
```

BARFI | Page No. | Date

Binary operator:

Arithmetic operator	Relational	Assignment	Logical
+	>	a = 10	&& || !
-	<		0 0 0 0 1
/	>=		0 1 0 1 0
%	<=		1 0 0 1 1
*	!=		1 1 1 1 0
10%2 = 0	==		
2%10 = 2	7<5		
	1 if true		
	0 if false		

→ (!(!1 && 5) && (1 && 1))
→ (0 && 1)
→ !0
→ 1

→ ("%d", 7<5) 10
→ ("%d", 8>5) 11

Type casting :-

conversation of one data type to another is called typecasting
→ it is of two types
a] implicit
b] explicit

q = integer
f = float.

q op q → q
q op f → F
f op q → f
f op f → f

a] implicit → int a=5
int b=2
int c=a/b;
("%d", c) //2

int a=5
int b=2
float c=a/b
("%f", c) //2.00

b] explicit → int a=5;
int b=2;
float c=a/b; → float c=(float)a/b;
("%f", c) //2.5

int num, rem1, rem2, rem3, sum;
pf("Enter 3 digit no");
sf("%3d", &num);
Pf("number is %d \n", num);
rem1 = num%10; //3
num = num/10; //2
rem2 = num%10 //2
num = num/10 //1
rem3 = num%10 //1
num = num/10 //0
sum = rem1 + rem2 + rem3;
pf("%d", sum); //6

pf("%d + %d + %d", rem3*10, rem2*10, rem1);

char ch='A'; //65 NULL → 0
char ch2=' '; //32
char ch3='\n'; //10
pf("ASCII value of %c is %d", ch, ch);

```
char ch='A';
pf("%c %d", ch+10, ch+10);
```

1] return type of scanf function is integer
2] It return How many i/p accepted from the user
3] The return type of pf funct is integer
4] It return How many character are enclosed within double quotes

Scanf

```
int a,b,c;
// int num = scanf("%d%d%d",&a,&b,&c);
pf("number of i/p is %d", num);

pf("%d", scanf("%d%d%d",&a,&b,&c));
```

Printf

```
#include<stdio.h>
int main()
{
 int x = Printf("Hello World");
 Printf("\n%d", x);

 Printf("%d", Printf("Hello World"));

 getchar();
}
```

integers
ted from the user
d integers
are enclosed within

,&c);
m);

,b,&c));

;

```
int a=10,b=90;
Pf("%d",printf("%d%d\n",a,b));
```

o/p:- 10 90 5

1090\n → 5
12345

* NULL, NULL
↓
'\0' → null character

NULL → predefined constant

```
printf("%d\n",NULL);
Pf("%d",'\0');
```

o/p:- 0
0

* Control Statements :-

The flow of the prg can be altered by using if, if else and Switch

```
if(condit^n/expression) → if true = 1
{
   == statements
}
```

```
if(1) // 10, -10, 1&&1, !0, 'A', 10>3, 'a'+1, 6+7
{
Printf("\n Hello");
}
if(0) // '\0', NULL, !6, 10&&0, 10<3, 9-8-1, ' '-32
{
   Pf("\n Hello");
}
```

```
if (0);
{
Printf("Hello");
}
```

if block will terminate but still Hello world will be printed ; as ';' means end of statement

★ if-else :-

there should not be any statement betn if & else Loop

```
if(1);
{
Pf("Hello");
}
else
{
 Pf("Bye");
}
```

```
if(1); (if ';' is not there then error)
else
{Pf("bye");
}
```

//Nothing will Print

it will give error as the syntax is voilated and the there cannot be statement betn if & else

★ Switch :-

Switch expression can Have integer and character variable name and constants, LE, RE, AE

2] Switch expression cannot have float and double variable name and constant, string, NULL,

3] case value can have int & chare constant, LE, AE, RE (Logical, Arthmatic & relational expression)

4] Cas value cannot have float & Double constant, String.

5] Case value should not have variable name
6] Case value should not be repeated

```
/* Switch(switch expression)
   {
   case casevalue1;
   case casevalue2;
   }
*/
```

```
if (0)
{
Pf("hi");
}
Switch(0);
{
case 0: Pf("Hello");
}
```

The Right value of comma is printed.

```
Switch(1,2)
{
case 1: Pf(Hi);
case 2: Pf(Bye);
}
```

→ use break keyword to take the control out of Switch case Loop.

```
case exp:
break;
```

→ Default is used when no matching case is found.

→ We can write the ASCII or any value as case no.

```
int a=65;
Switch(a)
{
case 'A': Pf("Hello");
}
```

1] int i=2;
switch(i)
case 4/i; pf("Hello");
o/p: error

2] switch(1);
case 1: pf("%d", (printf("hi")));
//error as there is semicolon after switch
Nawill print if ';' is not there.

3] num=1;
switch[num, num+1]
{
case 2: pf("Hi");
}
// Hi
// if printf("Hi")
o/p :- Hi2

4] switch(4>>5)
case 1 search;

5] switch('10').
{
case NULL:
pf("in case 0");
case 'lll':
pf("in case 1");
}
//in case 0
in case 1

6] switch(printf("hi"));
{
case 2: pf("Hello");
}
// Hi
Hello

```
int i=3;
Switch(i)
{
default : Pf("zero");          // three
case (1) : Pf("one");          if i= 60
break;                         then -> zero One
case (2) : Pf("two");
break;
case (3) : Pf("three");
break;
}
```

Types of variable:

1] Local
The variable declare within the braces of funtn or a block is called local variable
2] local variable is stored on stack section
3] intial value is grbage
4] Scope & Lifetime is within the block or within the function.
1] Scope defines How long the variable can be access
2] Lifetime defines How long the variable exists in the memory.
3] Lifetime of variable depends on Scope of variable
4] two Scope. can have same variable names if the scope are different

```
{
a=10;
Pf("%d",a);
{ a=20
Pf(a)          // 10
}              20
               10
```

3] Global variable
→ The variable declare above the Main is global
→ Global variable is stored on data section
→ Intial value is Zero
→ Scope & life time is throughout the prg.

Local is Having High priority over Gobal

Loops :-

when a certain task is need to be repeated No. of times loop should be used
2 types of Loop

Pre-tested Loop	Post-tested Loop
condition/exp is checked → false → Exit; True → State. inside Loop get exec (for Loop, while)	Statement inside Loop → exp^n is checked → false → Exit

Pointer :-

int a=10 → [10] · a→10→int
&a→1000→int*
a→int
1000→int*
int *b=&a; → [1000]
1000 (int*)
b→int*
int **c=&b; → 2000→int**
2000 (int**) → [2000]
c→int**

*a &a=1000

a→10→4
&a→1000→4
b→1000→4
&b→2000→4
*b→10→4
c→2000→4
**c→10→4

a == *b == **c

&a == b

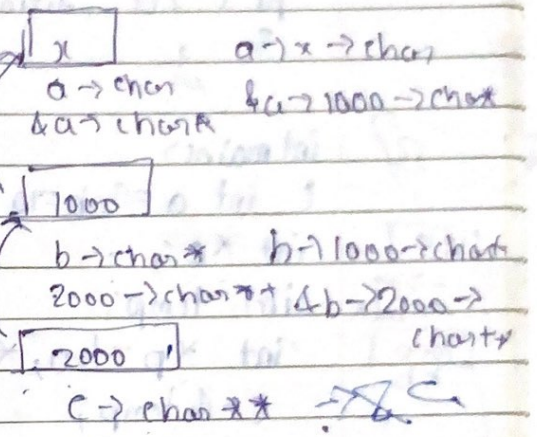
&b == c

```
char a='x';
char *b=&a;
```

1000 (char *)

```
char **c=&b
```

2000 (char **)



a → x → 1

&a → 1000 → 4

b → 1000 → 4

&b → 2000 → 4

*b → x → 1

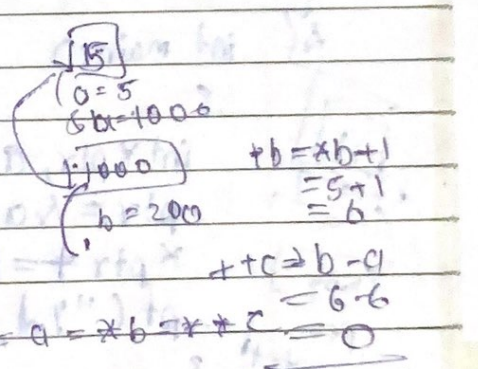
c → 2000 → 4

**c → x → 1

a == *b == **c

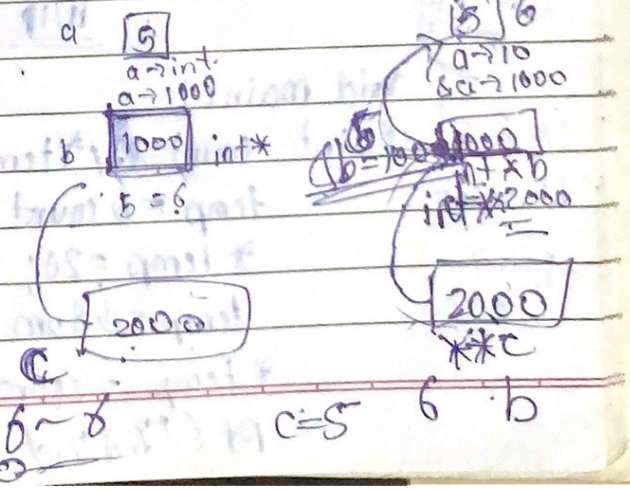
&a == b

&b == c



Code:-

```
int main()
{ int a=5,*b,**c;
  b=&a;
  (*b)++; // *b = *b + 1
  c=&b;
  **c = *b - a;
  pf("%d", **c);
}
```



BARFI Page No. Date

2) Void main()
{
double **ptr; // 8
char *ptr1; // 8
pf("%d", sizeof(ptr)+sizeof(ptr1)); // 9
}

3)
int main()
{ int a=5, b=10;
int *c;
int temp;
int *p=&a, *q=&b;
c=&temp;
temp = *p - *q;
pf("%d", *c);
}

4)
int main()
{
int *ptr, a=10;
ptr = &a;
*ptr += 1;
pf("%d, %d\n", *ptr, a);
}

// 11 11

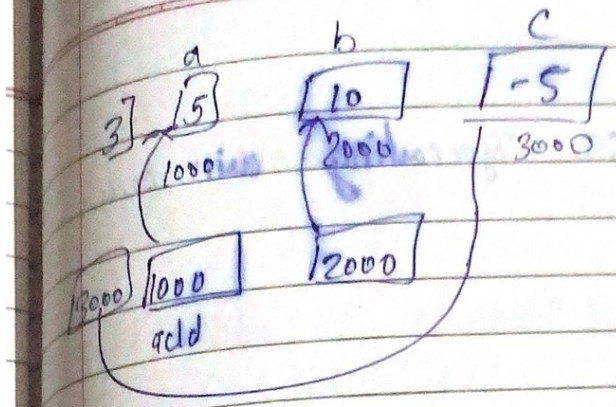
Void main()
{ int count=10, *temp, sum=0;
temp = &count; // 1000
*temp = 20; // 10 20
temp = ∑ // 2000
*temp = count // 10
pf("%d %d %d", count, *temp, sum); // 10, 10, 0

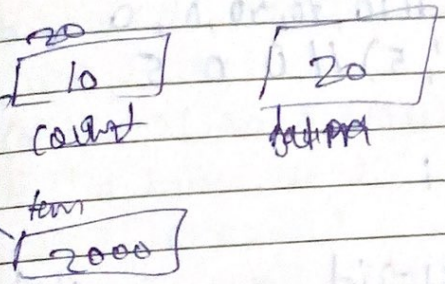
5)
void main()
{
int i=6, *p, **p1;
p = &i; // 1000
p1 = &p; // 2000
pf("%d", (*p+5/2));
}

6/2 = 3 // 1/2 = 5 => 6+2 = 8

6)
void main()
{
int i=3, *ptr, **ptr1;
ptr = &i; // 1000
ptr1 = &ptr; // 2000
*ptr = **ptr1 + i; // 6
**ptr1 = 1; // 1
pf("%d %d %d", *ptr, **ptr1, i);
}

// 5, 5, 5





20, 20, 20

Array :-

-> Array is set of Homogenous element in which elements are stored in a sequential manner

-> int a[5] = {1,2,3,4,5};
printf("%d", sizeof(a)); //20
printf("%d", sizeof(a[4])); //4
}

4 x 5 elements = 20

1	2	3	4	5
a[0]	a[1]	a[2]	a[3]	a[4]
100	104	108	112	116

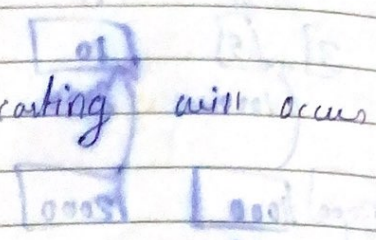
-> Name of the array itself is a pointer pointing to the base address of an array

a[0] = 100
a[0] = 1

a[5] = {1,2,3,4,5,6} -> compile time error

BARFI | Page No. | Date

```
int a[2]={'A'};
pf("%d",a[0]); //65  Typecasting will occur

int a[5]={1,2,3,4,5};
int a[]={1,2,3};
int a=10,b=70,c=40;
int a[5]={a,b,c} //10,70,40,0,0
int b[3]={NULL,'10',5) //0 0 5
int size=5;
int a[size]//error;
```



```
int a[5];   a[0] → 1 → int
            &a[0] → 100 → int *
            int * a = &a[0];
                  (100(int *)).
```

1|2|3|4|5
a 100 104 108 112 116

int * a
100

Address

&[a] = 100
&a[i] = 104
&a[2] = 108

(Name of array | Point + index)

a+0 → 100 (0+a)
a+1 → 104 (1+a)

(a+0) → 100
&a[0]
(a+0)
(0+a)
a

Address 100

∴ &a[i] == (a+i) == (i+a)

Value

a[0] → 1 a[a]
a[1] → 2 1[a]

* (Name of array | Point + index)

* (a+0) → 1 * (1+a)
* (a+1) → 2 * (1+a)

*a
a[0]
*(a+i)
*(i+a)
0[a]

(Value)

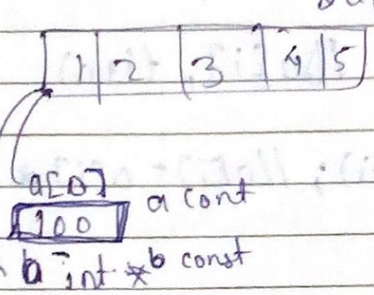
∴ a[i] = i[a] == *(a+i) = *(i+a)

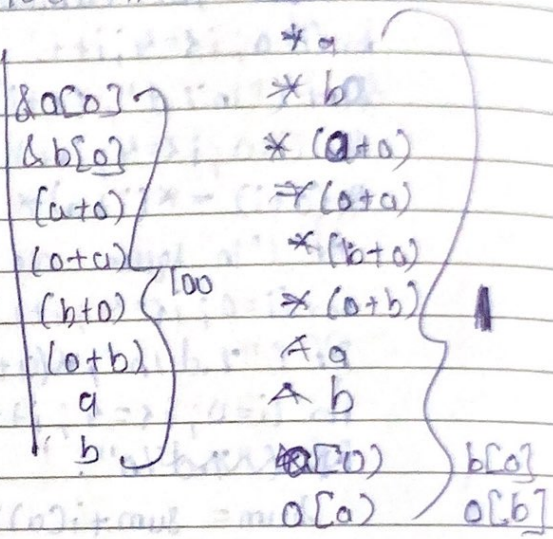
```
int a[5];
int sum, avg, i;
float avg;

Pf("Enter the elements");
for(i=0; i<=4; i++)
scanf("%d", (a+i)); //&a[i]; (i+a)
for(i=0; i<=4; i++)
*(a+i) = *(i+a) * *(a+i); //a[i] = a[i] * a[i];
Pf("In square no");
for(i=0; i<=4; i++)
Pf("%d\n", *(a+i));
for(i=0; i<=4; i++)
Pf("avg\n");
    sum = sum + i(a);
    avg = (float) sum/5;
    Pf("avg %f \n", avg);


int a[5] = {1,2,3,4,5};
 int max;=
   max=*(a+0)
   for(i=0; i<=4; i++)
   {
     if (max>a[i])  (max<=*(a+0)*(a+i)
     {
        max=a[i]+i;   max = *(i+a);
     }
     else
```

max = 0
i < a[i]
max = 1
i = 1

Pointer +/- constant = memory location (vice versa)
Name of array is constant pointer pointing to have address of array

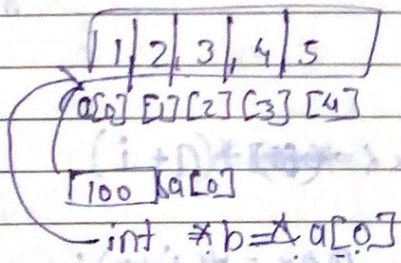
```
int a[5];
int *b = a;  // (a+0) (0+a) &a[0]  } 100 -> int*
```



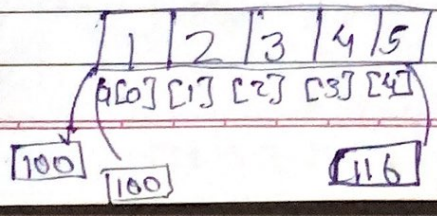


Arthmatic operation on pointer :-

1] increment a pointer
2] Decrement a pointer
3] Adding integer to pointer
4] Subtracting Integer from pointer



5] Subtraction of two pointer



// code

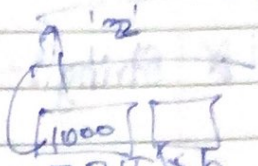
```
int a[5] = {1,2,3,4,5};
int *b = &a[0];
// a++ error it is constant
b++ // b+=1 // b = (100+1)
b-- // b-=1 // b = (100-1)
b+=3 // b = b+3 (112)
b-=2 // b = b-2 (104)
int a[5]={1,2,3,4
int *b=&a[0];
int *c=(a+4)
```

100 + 10 = 200

#P

in

int a[5] = {1,2,3,4,5};
int *b = &a[0];
int *c = (a+4);



Pf("%d", c-b) // ⇒ 116-100

(100 / 4)

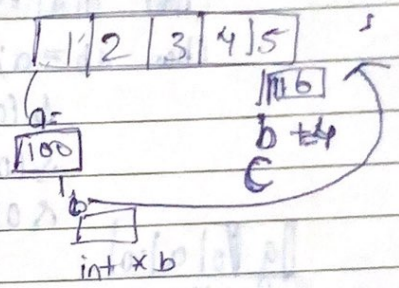
16/4 int = 4

4 element far from a & b

6] Comparing two pointers

7] Assinging one pointer to another pointer

int *b = &a[0];



The following operation cannot be done on pointer

1] multiply two pointers
2] Div two pointer
3] Addition of two pointer.

* Note :-

Ptr - ptr → no of elements
pointer ± const → mem loc
Pointer *, / · const → not allowed
pointer +, -, * ptr → not allowed.
*value ± value = value (According to Natural eq

1	2	3	4	5
a[0]	[1]	[2]	[3]	[4]
100	104	108	112	116

int a[5] = {1,2,3,4,5};
pf("%d\n", a[4]+2); // 5+2 → 7
pf("%d\n", &a[4]-2); // 116-2-2 → 3

String :-

Character Array	String
char a[4] = {'g', 'o', 'o', '\0'};	char a[4] = "goo";
g o o \0 → user should give	g o o \0 implicitly occurs
100 101 102 103	

```
char a[4] = "goo";
char *b = a;
        &b[a+0];
        &[0+a];
        &a[0];
```

(char *) 100

g o o \0
100 101 102 103
a[0] a[1] a[2] a[3]
100 100
a b (const char *)

a[0] → g → char
&a[0] → 100 → char *
char *a = &a[0]
↓ 100 (char *)

&a[0]
&b[0]
*(a+0)
*(0+a)
a
b
(0+b)
(b+0)

100

*a
*b
a[0]
a[a]
b[0]
a[b]
*(a+0)
*(0+a)
*(b+0)
*(0+b)

g

// code

```
char a[4];
printf("Enter String");
scanf("%s", a); //(a+0), (0+a), &a[0] ---> goa
pf("String is %s \n", a); // (a+0), (0+a), &a[0] ---> goa

sf("%s", a+1); //&a[1], (1+a) ---> go
Pf("String is %s \n", a); // ---- goabag go

sf("%s", a); //goa
Pf("String is %s \n", a+1); //oa
pf("%c", *a); //g
```

%s → Accepts only one string, & if two str it will not Hold.

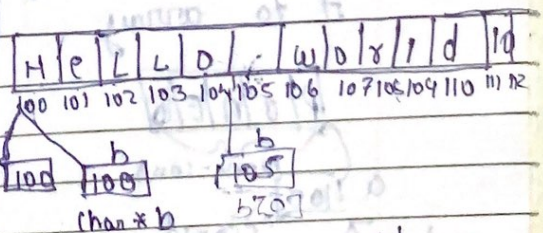
// code

```
Char a[20];
Pf("Enter multi String");
gets(a);
puts(a);
```

// Code

```
char a[12] = "hello world";
char *b = (a+a);
b+=5;
Pf("%s \n", b);
Pf("%s \n", a);
Pf("%c \n", *b);
Pf("%c \n", *a);
pf("%c \n", b[0]+2); ---> b[1] => w, x, y  //y
Pf("%s \n", b+2);
```

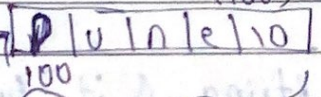
word Pf("%s \n", b+1);



Char a[12] = "hello"; //12
~~printf("%d", Strlen(a);~~ //15
Char a[] = "hello "; //16
Char a[12] = "hello\n"; //12
Char a[] = "hello\n"; //7

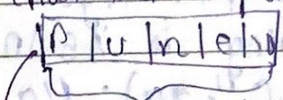
Strlen() → it counts the length of String excluding '\0'

Char a[5] = "pune";
returns base address (100) → char *



Non-constant a++; ✗
a+1; ✓
a → constant char * a[3] = a; ✓

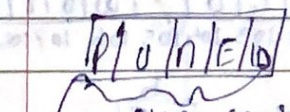
Char *ci = "pune";
return base address (100) → non-const



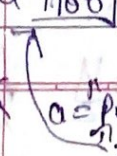
constant
Non-constant
ci++; ✓
ci+1; ✓
ci[3] = 'a'; ✗

Imp Note

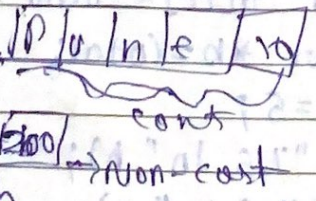
Char a[5];
a = "pune";
→ This is not allowed as this a array and its constant so we cannot assign it to array



Can't go ✗



Char *b;
b = "Pune";
→ This a Non-constant array so we can assign it



```
// code
int cnt=0;
char a[6] = "pune";
char *b=a;
for(;*b!='\0';b++)
  {
   cnt++;
  }
Printf("\n%d", cnt);
```

```
Function()
int i;
for(i=0;a[i]!='\0';i++
  {
   cnt++;
  }

while(*b!='\0')
  {
   b++;
   cnt++;
  }
```

* Structure →
it is set of Hetrogeneous set of datatype in which elements are stored one after another
* during declaration of structure mem allocation do not take place for the members of structure
* mem is allocated at the time of variable creation of structure. →

```
#include<stdio.h>
Struct employee
{
  int id;
  float sal;
  char name[10];
};
int main()
 {
  Struct employee e1, e2;
  Pf("accept elments");
  Sf("%d %f %s", &e1.id, &e1.sal, &e1.name);
  Pf("id is = %d \t sal is = %f \t name = %s \t", e1.id, e1.s
```

Structure variable
pointer →
[].
() →
Scanf(&)