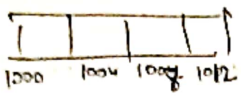
Array

int a[ ]; [ ] one operator — Array Index Operator.

↓ Unsigned Integer value

Base Address → address of the 1st element of the array

Byte Addressable



1int = 4Byte

1000 → 1 Byte
1001 → 1 Byte
1002 → 1 Byte
1003 → 1 Byte
} To store 1 integer.

int a[4] = {1,2,3,4};

↳ size can be skipped, if Initialization is done

int a[4] = {1,2,3,4,5,6};

↳ only 4 element will fit

Default storage Local array = stack

global array = Data segment

int a[4] = {1,2,3,4};

1000	1004	1008	1012
1	2	3	4

a is the Name of array

printf("%u", a);

→ 1000

a represent address of 1st elemet of array

a can't be modified or updated

Pointer Arithmetic → depends upon the size of datatype

int a[4] =

1	40	12	14

Addition

```
int *ptr;
ptr = a;
ptr++; -> ptr = ptr+1 -> ptr = ptr + 1 * sizeof datatype
```

If datatype is int → ptr = ptr + 1 * 4

If char, ptr = ptr+1 * 1

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

for int

$a[0] = *(a+0) = *(1000+0) = *(1000) = \_\_$

$a[1] = *(a+1) = *(1000+1) = *(1004) = \_\_$

(1004 ← Size of datatype)

```
#include <stdio.h>
int main() {
    int a[] = {11, 12, 13, 14};
    printf("%p\n", a);
    printf("%p\n", &a);
    printf("%p\n", &a[0]);
    return 0;
}
```

11	12	13	14
100	104	108	112

a → 100 → Integer Address.

a → address of first element of Array

&a → Address of one Dimensional Array

→ will print 100

But array of integers

"%p" → pointer's address

Hexadexima!

$\&a[0] = \&(*(a+0))$

$\&(*100) \rightarrow \&(11)$

↓

100

"%p" $a+1 \rightarrow 104$

$\&a+1 \rightarrow 100 + 1 \times$ size of array $= 100 + 16 = 116$

$\&a[0]+1 \rightarrow \&(*(a+0))+1 \times$ size × datatype.

$\&(*(100))+1$ ⁱˣ — —

$\&(11)+1$

$100 + 1 \times$ size of datatype $= 100+4 = 104.$

(with size of array)

Address

↓

pointer$_2$ − pointer$_1$

Size of datatype

```
int a[] = {10, 20, 30, 40, 50};
int i, *b, *b1;
b = a          // 1000 is assigned.
b1 = a+4       // 1000 + 4×4 = 1016
i = b1 - b;    → (b1 − b) / Size = (1016 − 1000) / 4 = 4
printf("%d", i); → 4
```

#

```
int a[] = {1, 23, 4, 5};
int *b = a+3;
printf("%d", b[-2]);
```

100 104 108 112

→ 100 + 3×4 = 112.

*b = 5
b = 112.

b[-2] = 112 − 2×4
= 112 − 8
= 104

*(b-2)

↓
23.

2 D Array:

```
int a[2][3] = {{1, 2, 3}
               {4, 5, 6}};
```

In memory

1	2	3	4	5	6
100	104	108	112	116	120

Sequential

a → Add. of 1D array

*a

**a.

&a → Address of 2D array

a[i][j]

↓

*(a+i)[j]

((a+1)+j)

```
int a[2][2] = {{11, 12}, {13, 14}};
```

100 104 108 112

a → 100

&a → 100

*a → 100

**a = 11

Size of 1D array

a+1 → 100 + 1×8 = 108

*a+1 → 100 + 1×4 = 104

**a+1 → *(*(a+1))

*(104) → *(12) = 13

**(100+4) → 12.

&a+1 → 100 + 1 × Size of 2D Array

100 + 1×16 = 116.

Starting add. 1002, size of element = 2B.

```
int a[3][4] = {{1, 2, 3, 4}
               {5, 6, 7, 8}
               {9, 10, 11, 12}};
```

```
a[0]+1, *(a[0]+1), **(*(a+0)+1);
```

a[0]+1 → *(a+0) + 1

*(1002) + 1

1002 + 1×2

1004

((a+0)+1)

*(1002+1)

*(1002+1×2)

*(1004)

↓

2

Declaration.

int a[ ][ ]; ✗

int a[4][ ]; ✗

int a[ ][4]; ✗

int a[4][3] → in declaration.

int a[ ][ ] = {{1,2},{3,4}} ✓

✓ int a[ ][2] = {60,70,80,40}

✓ int a[2][2] =

String

↓

Array of character.

'\0' → ASCII value 0

'\0' → null character.

String's last alphabet is '\0' (zero) → where string ends.

char ch[ ] = "String";

Name of array represent address of 1st element of array.

printf("%s", ch); → this must be address '%s'

↳ till '\0'.

ch[ ] = "a\0"

%s, ch → a

ch[1] = 'p'

↳ Allowed

① ch = ch+1 → error updation not allowed.

② ch = "spring" → not allowed.

Second way of declaring string using character pointer

char *ptr = "string"

ptr [100]

ptr = ptr+1 → is allowed.

ptr[1] = 'p' → Not allowed

char a[ ] = "string";	char *ptr = "string";
a = a+1 → ✗	ptr = ptr+1 ✓
a[2] = 'p' → ✓	ptr[2] = 'p' → ✗

char ch1[] = "string"

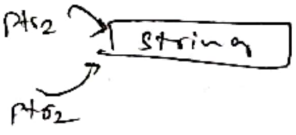
char ch2[] = "string"

ch1 → [string]

ch2 → [string]

char *ptr1 = "string"

char *ptr2 = "string"



Array of String.

char a[2][10] = {"parakram", "vijay"}

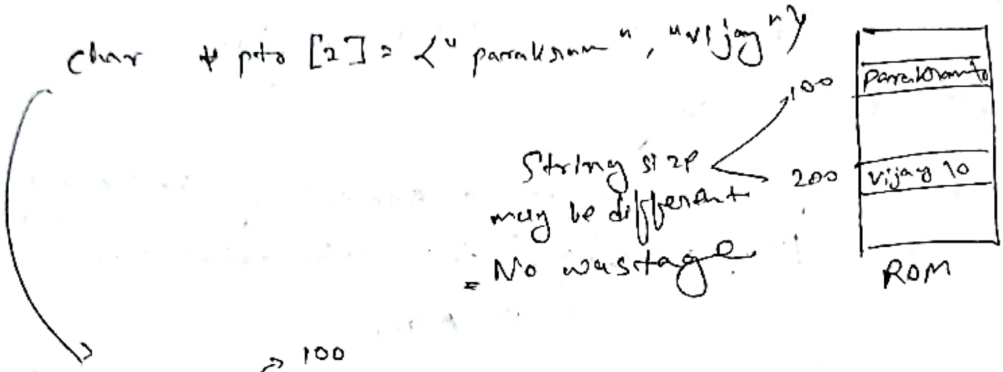
→ wastage of space

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
p	a	r	a	k	r	a	m	\0	\0	v	i	j	a	y	\0	?	?	?	?

String character Pointer

Array of character pointers → char *ptr = "parakram";
char *ptr1 = "vijay";

char *ptr[2] = {"parakram", "vijay"}



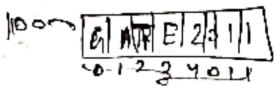
%s, ptr → parakram (ptr = 100)

%s, *(ptr+1) → *(100+1) → 101 → arakram (address till \0)

%s, *(ptr+1) → *(100+1×10) = *(110) = vijay\0 (10 Array)

GATE. 2011

```
Char C[] = "GATE2011"
Char *p = C;
printf("%d", p[3] - p[1]);
```



value, value

100 + *(p+3) *(p+1)
*(100+3) *(100+1)
103 - 101

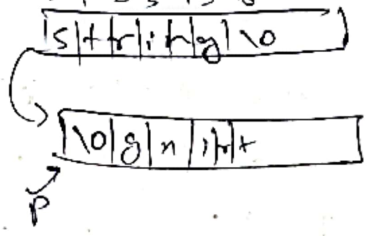
103 → E ASCII → 69

101 → A → 65

2011 = 100 + 69 − 65 → 104

GATE 2004.

```
char p[20];
char *s = "Stolen";
int length = strlen(s);  → 6
for (i = 0; i < length; i++)
    p[i] = s[length - i];
printf("%s", p);
```



→ %s till \0 but 1st element is \0 ∴ No output

char c[] = "GATEWALLAH"
100 1 2 3 4 5 6 7 8 9

char *p = c;

printf("%s", p + p[8] - p[1]);

100 + *(100+8) - *(100+1)

100 + *(108) - *(101)

100 + 'O' → GATEWALLAH

Strcmp

int Strcmp(const char *ptr, const char *ptr1)

2 string are being compared

if string are equal return → 0

If 2 string are $S_1 < S_2$ then less than 0

" " " $S_1 > S_2$ " more than 0.

Strcmp("Abc", "abc") → -1

65 < 97

("Abcdef", "abc") → -1

65 < 97

("abcd", "abc") → 1

("abe", "abcdef") → 1

Strcpy

strcpy (char * destination, const char *ptr)

char a[20];
strcpy(a, "xyz")

Ch[20] = "parakram";
strcpy(ch, "vijay");
printf("%s", ch) → vijay

parakram\0

vijay\0amb

strcat

char * strcat(char *des, const char *src)

parakram\0
vijay\0

vijay\0

2017

0 1 2 3 4 5 6 7 8 9 10 11

Char *c = "GATE CSIT 2017";

Char *p = c;

Printf("%d", (int) strlen(c + 2[p] - 6[p] - 1));

0 + *(p+2) - *(p+6) - 1 (Pattern 'x')

Address 0 + Ascii 'T' 84 - 73 - 1 x Pattern

0 + 11 - 1 x 1

10 → Address

↳ "17" → 2 length.

For strlen
↳ unsigned
"%u"

Structure

↳ collection of disimilar data type.

(User defined Data type)

```
Struct A {
    int a;
    char ch;
};

Struct A var;
  Name of Structure ; var ← Structure variable

var.a = 10;
var.ch = 'A';
```

Struct A var = {10, 'A'};

Create Alias

```
typedef Struct S {
    char *name;
    int Roll no;
} Student;

Student var;
```

Structure pointer

```
Struct S S1 = {"Vi", 23};
Struct * ptr;
Struct * ptr = &S1; // ptr = &S1;
```

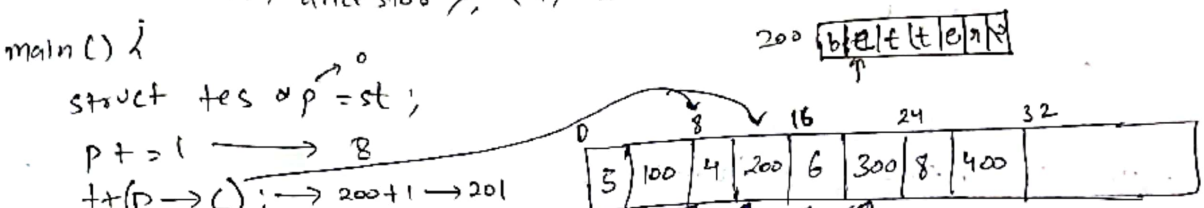
How to access member of a structure using structure pointer.

"→" arrow operator.

```
Printf ("%s", ptr→name);
printf ("%s", (*p).name);
```

```
# struct tes {
    int i;
    char *ch;
} st[5] = {{5, "become"}, {4, "better"}, {6, "jungle"}
           {8, "ancestor"}, {7, "brother"}};

main() {
    struct tes *p = st;
    p += 1 ⟶ 8
    ++(p→c); ⟶ 200+1 ⟶ 201
    printf ("%s", p++→c); ⟶ etter
    printf ("%c", *++p→c); ⟶
}
```



++(p→c) → 300 → 301 → u

more precedence

Union

It allows to use same memory location for diff. data type

```
Union Data {
    int i; ⟶ 4B
    float f; ⟶ 4B
    char str[32]; ⟶ 32 bytes
};
```

[Location size is equal to maximum memory required.]

alingment constraints

New variable must be divisibly by 4

```
Struct temp {
    int i; ⟶ 4B
    char ch; ⟶ 1B
} ⟶ 5B
```

Every pointer (any type) 4B

#define Macro

Language Preprocessor.

→ Macro expansion.

```
#define Max 100   ← No semicolon.
if (size == Max)  → if (size == 100)
```

Parameterized macros

```
#define Max(x, y)  x>y ? x : y
```

```
#define square(x)  x*x
int main() {
    int a, b = 3;
    a = square(b+2);
    printf("%d", a);
    return 0;
}
```

→ because preprocessor will blindly execute

b+2 * b+2

3+2 * 3+2 → 11

(b)

Dynamic Memory allocation.

↳ In Heap memory (Runtime)

→ `#include <stdlib.h>`

malloc() ⇒ ~~All location assigned to garbage value~~ Memory is not initialized.

↳ single parameter, size of memory allocated

↳ malloc void pointer return.

```
int *ptr;
ptr = malloc(sizeof(int));
```

→ `ptr = (int *) malloc(sizeof(int));`

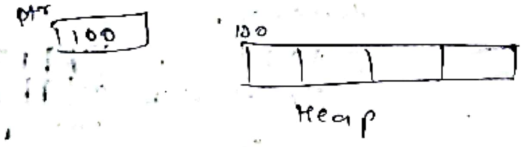
type casting

↳ Its not mandatory as type cast is automatically

of 4 byte

* Returns void pointer

∴ No error




```
ptr = malloc(10 * sizeof(int))
```

void *ptr → void pointer can't be dereference.

Void pointer, pointer arithmetic does not work.

Calloc() → contiguous allocation. All location assigned to zero.

↳ 2 parameter.

Calloc(no. of location, size of(datatype))

Calloc(75, sizeof(char)); ← 75 bytes.

Deallocation is also important.

↳ free (address)

```
int *ptr = malloc (sizeof (int));
   :
free(ptr);  ⟶ if not deallocated,
                ↳ Memory Leak!
```

int *ptr = malloc (sizeof (int)) ⟶ [100] ptr ⟶ memory Leak.

ptr = malloc (sizeof (int)) ⟶ [200] ptr

free (ptr)

getchar & putchars

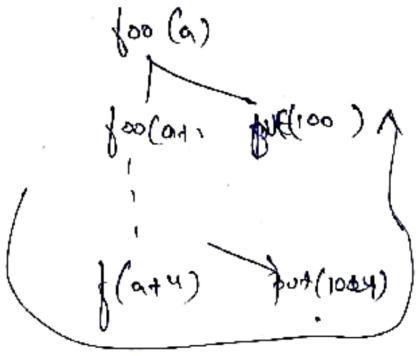
1 char, input getchar ()

putchar () :- output of 1 character in terminal.

"A(100) B C D(104) E F G H"

```
void foo (char *a){
    if (*a && *a != ' '){
        foo (a+1)
        putchar (*a);
    }
}
```

DCBA



1. Modulus operator works only with integers.

2. a = 3>2 ? 0 ? 0 : 1 : 5   2>3 ? 0 ? 0 : 1 : 5

a = 0 ? 0 : 1

a = 1

3. a=21
printf("%d", a++);

→ 21

4. scanf() → stops as encounters white space

getc() → ignores white space and scan till end, until finds new-line.

5. fun(-) {

return n--;

} ↑ return n then decrement.

6. int a = 3;
int c = sizeof(a++);

value of a will not change & remain 3.

7. % → mod operators only on integer else error.

8. int a = 3.

if(a++) | if(a<<2)

↑ value change | value not change after execution.

9. int x = -2024.

Printf("%d", ~(x+5)); → -2024+5 = -2019 (x)

Printf("%d", ~(x+1));

~(-2019) = -(-2019+1) = 2018

~(-2019+1) = ~(-2018) = -(-2018+1) = 2017

10. if(a = a<<1) → value inc. → Not short circuit

if(a = a++) → short circuit

↑ value don't inc.

if(a = a && b++) → short circuit

↳ In this case not happening but in case of || or it will happen