C Programming Lecture 9

Tuesday, 18 June 2024 8:18 PM

0x100 0x101 ... 0x109 0x10A 0x10B ... 0x10E
0x110 0x111

Pointer arithmetic :-

```
int *p, *q;
q = p + 5;
printf("%d", q-p);
```

p → 100

q = p + 5; → 120

size of (int) = 4

P = 0x100 → 100

q = 0x114 → 120

q - p = (120 - 100) / sizeof(int) = 5

```
int *p;
p = p + 5;
```

p → 0x100

100 + 5 (sizeof (int))

E ← 15
10 ← 16
11 ← 17
⋮
14 ← 20

Arrays in C

- Sequential collection of elements of same data type.
- As they are stored sequentially in memory, it gives the advantage to access any element in constant time using the index.

For a type **T**, **T[size]** is the type, **"array of size elements of type T"**.

The elements are indexed from 0 to size-1.

For example:

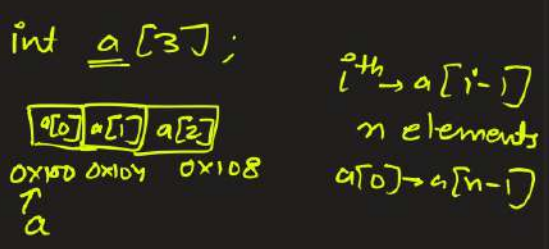
```
int arr[2];     // an array of two integers: arr[0], arr[1]
char* p[4];     // an array of four pointers to char: p[0], p[1], p[2], p[3]
```

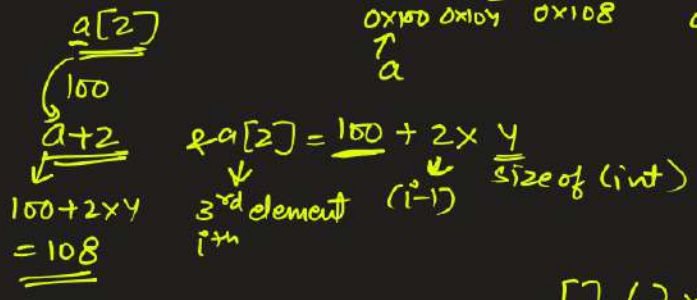
An array can be initialized by a list of values: ✓

- int arr[] = {4, 3, 2, 1}; ✓
- char p[] = {'x', 'y', 'z'}; ✓

Example: Correct or incorrect:

1. int arr1[3] = {1, 2, 3, 4}; ✓
2. int arr2[6] = {1, 2, 3, 4}; ✓
3. int arr3[6] = arr2; ✗

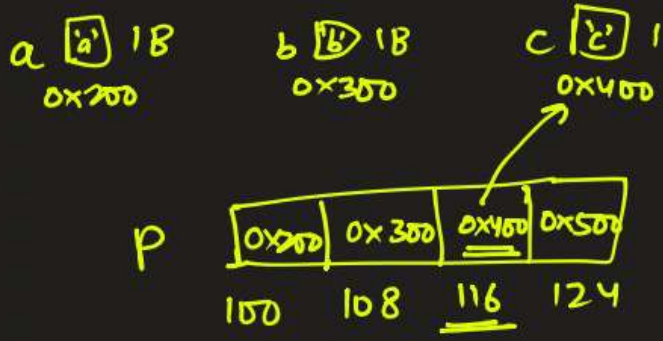




char a, b, c, d;

char * p[4];

p is an array of 4 character pointers



sizeof (pointer) = 8

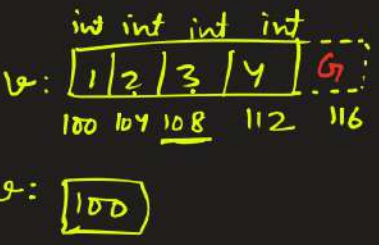
p[0] = &a, p[1] = &b, p[2] = &c, p[3] = &d

*p[2] = 'c'

Pointers and Arrays

The name of an array can be used as a pointer to its initial elements,

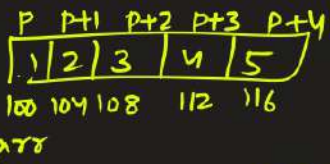
```
int v[] = {1, 2, 3, 4};
int* p1 = v;              // pointer to initial element
int* p2 = &v[0];          // pointer to initial element
int* p3 = v+4;
```




```
#include<stdio.h>
void main(){
    int arr[5] = {1, 2, 3, 4, 5};
    int*p = arr;
    for(int i = 0; i< 5; i++)
        printf("%d  ",*(p+i));
}
```

1 2 3 4 5

*p, *(p+1)
*(p+2) *(p+3)
*(p+4)



p 100 int *

sizeof(int) = 4B

int * 100 P1 8B

int * 116 P3 8B

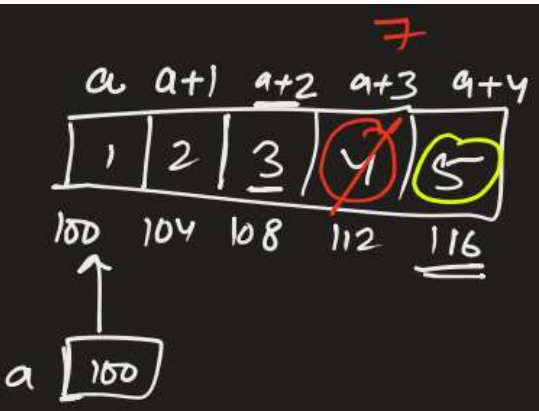
int * 100 P2 8B

*(p2+2) = 3

*(p3 + (p1 - p2) - 2) = 3

*(p3-2) = 3

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



a[2] ≡ *(a+2)

a[i] ≡ *(a+i)

printf("%d", a[4]);

↓ → 5

*(a+4)

a[0] = *(a+0)

= *a

a[3] = 7

*(a+3) = 7

&3 = 108

&(*108)

&a[2] = &(*(a+2))

= 108

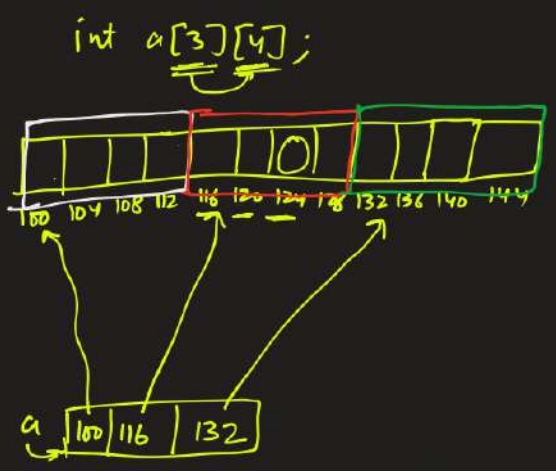
= a+2 = 108

printf("%d", a[i]);

Multidimensional Arrays

Arrays of arrays are the multidimensional arrays

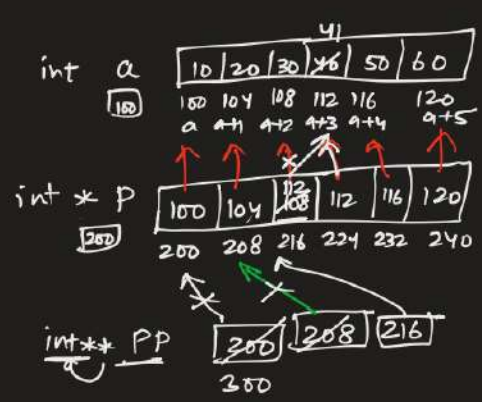
```
int marr[3][5];    // 3 arrays with 5 elements each

int arr[3][2] = {{0,1}, {2,3}, {4,5}};
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++)
        printf("%d", arr[i][j]);
}
```



eg:

```
int main() {
    int a[] = {10,20,30,40,50,60};
    int *p[] = {a, a+1, a+2, a+3, a+4, a+5};
    int **pp = p;
    pp++;
    printf("%d, %d, %d", pp-p, *pp-a, **pp);
    *pp++;
    printf("%d, %d, %d", pp-p, *pp-a, **pp);
    ++*pp;
    printf("%d, %d, %d", pp-p, *pp-a, **pp);
    ++**pp;
    printf("%d, %d, %d", pp-p, *pp-a, **pp);
    return 0;
}
```

output: 1, 1, 20 / 2, 2, 30 / 2, 3, 40 / 2, 3, 41



pp++;
pp = pp + 1
= 200 + 1 x sizeof(int*)

right to left

post inc/dec > pre inc/de = *

Example (GATE CS 2024 Set 2)

What is the output of the following C program?

```
#include < stdio.h >
int main() {
double a[2]={20.0, 25.0}, *p, *q;
p = a;
q = p + 1;
printf("%d,%d", (int)(q - p), (int)(*q - *p));
return 0;}
```

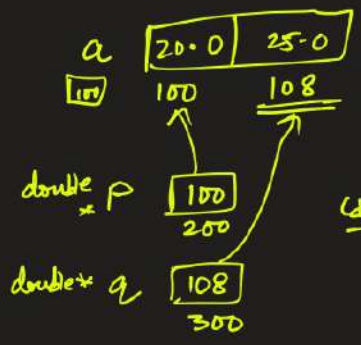
(1) (5)

A 4,8

B 1,5

C 8,5

D 1,8



$q - p$ (double) (double) / sizeof(double)

$= \frac{108 - 100}{8}$

$= 1$

$*q - *p$

$= 25.0 - 20.0$

$= 5.0$

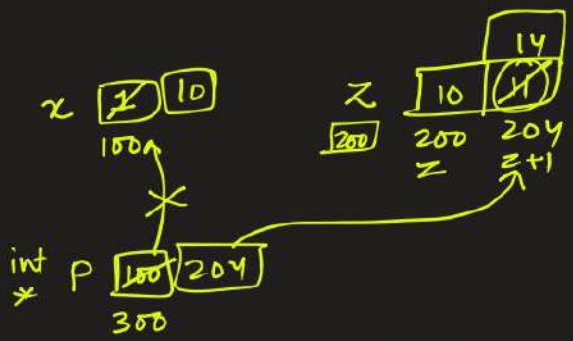
$int(5.0) = 5$

Example (GATE CS 2022)

What is printed by the following ANSI C program?

```
#include < stdio.h >
int main(int argc, char *argv[])
{
int x = 1, z[2] = {10, 11};
int *p=NULL; p=&x;
*p=10;
p =&z[1];
*(&z[0]+1)+=3;
printf("%d, %d, %d \n",x,z[0],z[1]); return   0;
}
```

10, 10, 14



&z[1]
= &(*(z+1))
= z+1

&z[0]
200+1
↓
*204

A. 1, 10, 11

B. 1, 10, 14

C. 10, 14, 11

D. 10, 10, 14

Multidimensional Arrays

Arrays of arrays are the multidimensional arrays

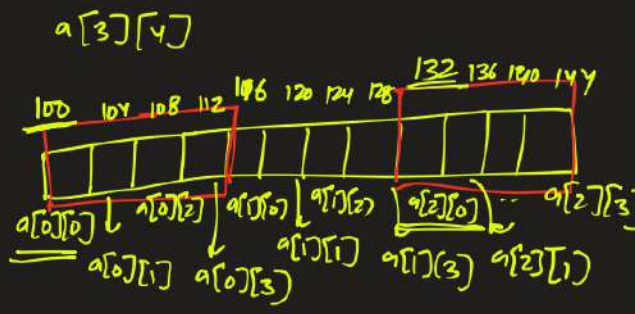
```
int marr[3][5];    // 3 arrays with 5 elements each
```

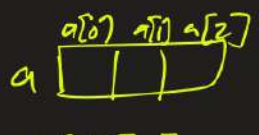
```
int arr[3][2] = {{0,1}, {2,3}, {4,5}};
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++)
        printf("%d, ", arr[i][j]);
}
```

rows

columns

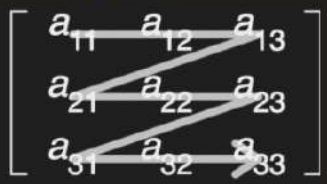
0, 1, 2, 3, 4, 5





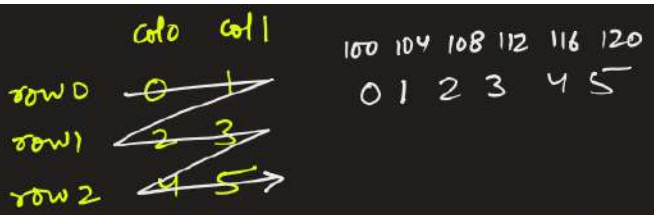
There are two methods of storing multidimensional arrays:

Row major order: The elements are arranged consecutively along the row.



Address	0	1	2	3	4	5	6	7	8
Element	a_{11}	a_{12}	a_{13}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}

```
Address[row][column] = Base address + row*num_columns*element_size + column*element_size
```



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

rows = 3

columns = 4

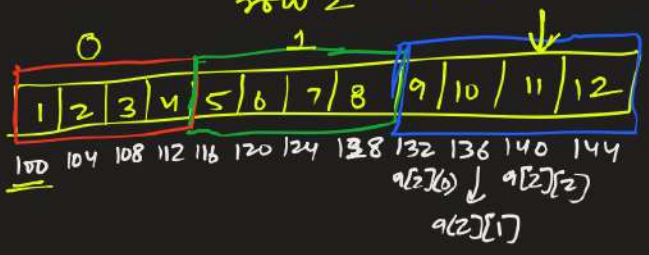
every row contained 4 columns

Base address (BA) = 100

address a[2][2] = ?

a:

	Col 0	Col 1	Col 2	Col 3
row 0	1	2	3	4
row 1	5	6	7	8
row 2	9	10	11	12



2 rows completely passed

in the 3rd row, 2 elements passed and we are at 3rd element

Total elements passed = 2 × 4 + 2 = 10

2 = # rows passed

4 = # columns

2 = # columns passed in 3rd row

Example (GATE CS 2002):

Consider the following declaration of a 'two-dimensional array in C:

row-major order

```
char a[100][100];
```

100 rows, 100 columns

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is

a) 4040
b) 4050
c) 5040
d) 5050

address of a[40][50] = BA + (40 × 100 + 50) * 1
= 0 + 4000 + 50
= 4050

Column major order: The elements are arranged consecutively along the column.

Address	0	1	2	3	4	5	6	7	8
Element	a_{11}	a_{21}	a_{31}	a_{12}	a_{22}	a_{32}	a_{13}	a_{23}	a_{33}

```
Address[row][column] = Base address + column*num_rows*element_size + row*element_size
```

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

rows ↘ columns

Find the address of a[2][2]

row column

elements passed

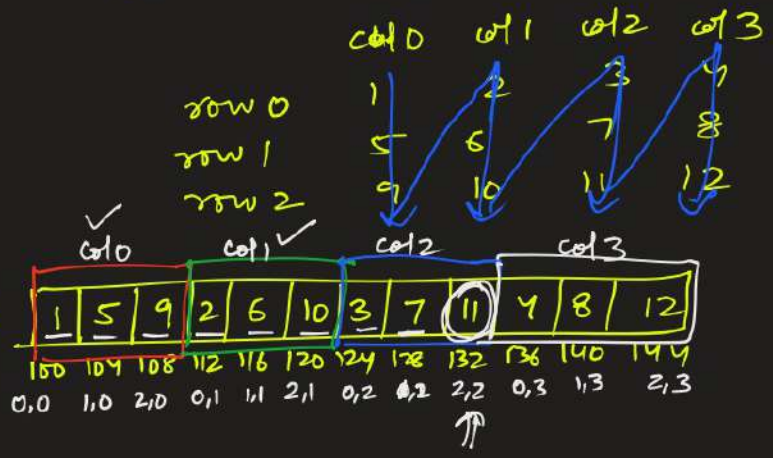
= 2 columns * 3

+ 2 rows in column '2'

= 8

∴ Address of a[2][2] = 100 + 8 × 4

= 132



Example [H/W]

Consider a two-dimensional array A with 3 rows and 4 columns stored in memory in column-major order. Given the base address of the array A is 2000, and indices i = 1 (row index) and j = 2 (column index), what is the address of the element A[1][2]? (Assume the array stores integers which hold 4 B of memory)