Windows Programming

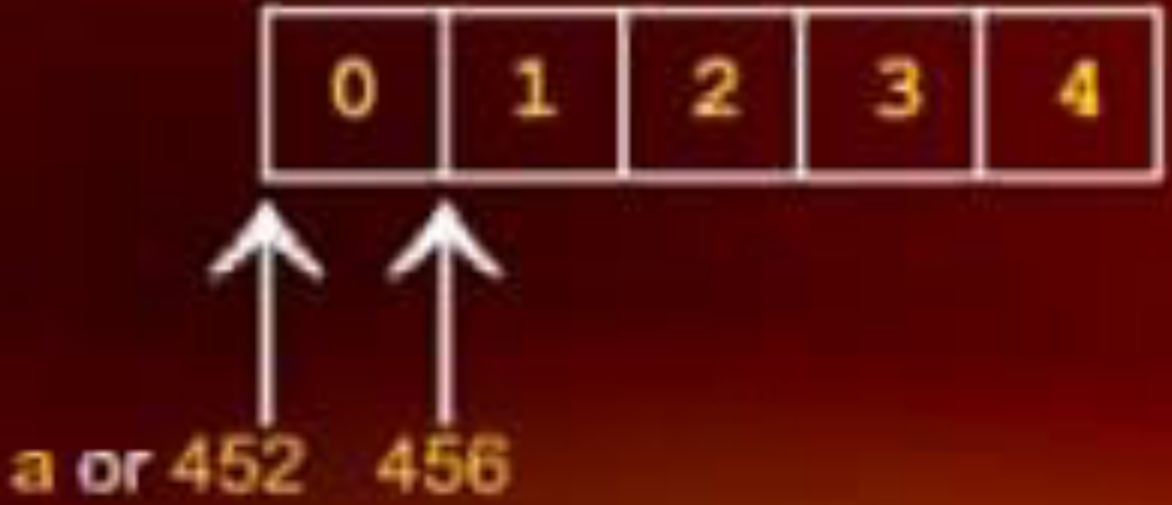
Lecture 03

Pointers and Arrays

Arrays

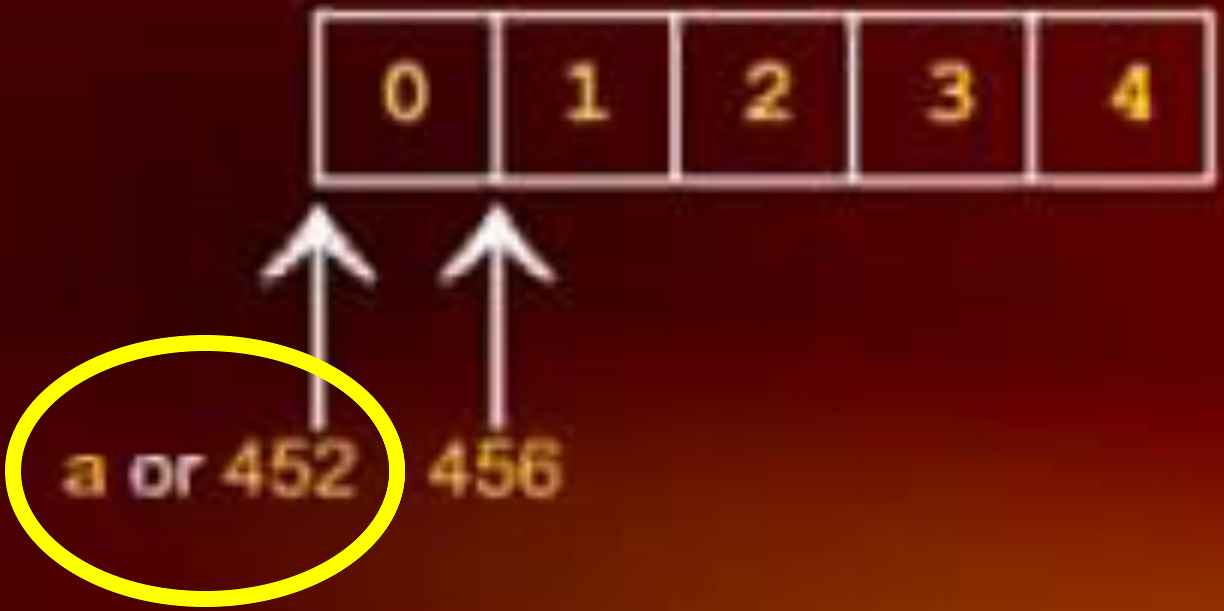
An array is a collection of variables of the same type. Individual array elements are identified by an integer index. In C the index begins at zero and is always written inside square brackets.

```
int a[5];
```



- According to C language specification,
  - Single dimensional array name is the starting address of array’s first byte.

```
int a[5];
```

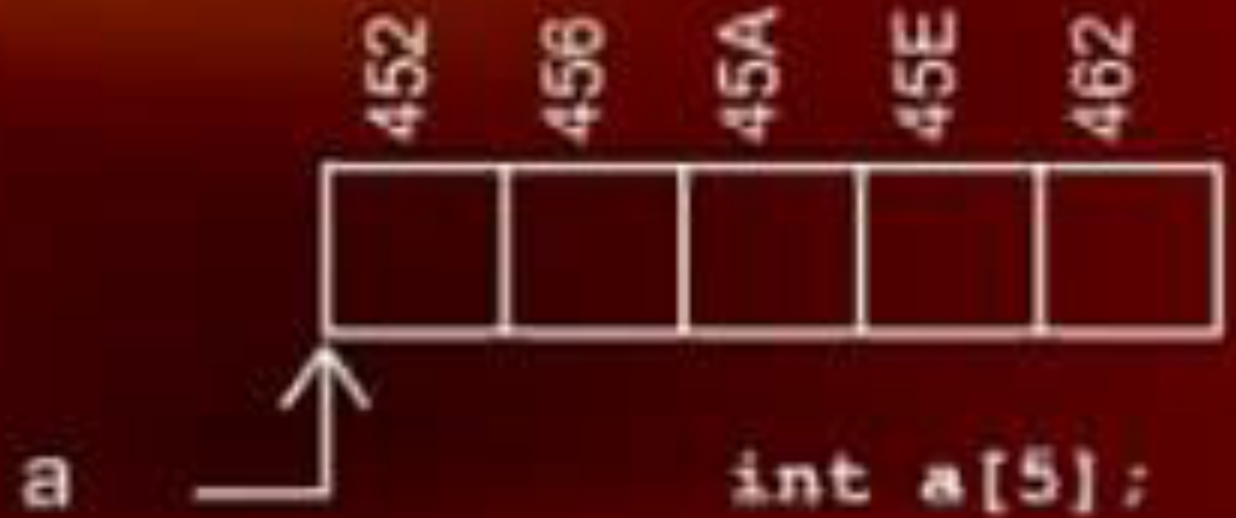


All arrays are

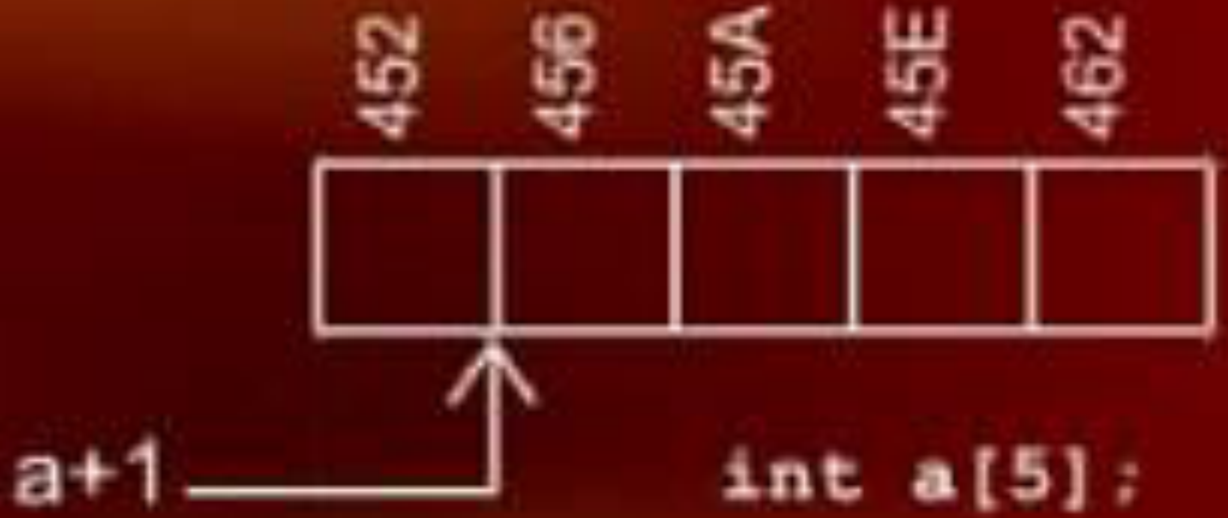
Zero

based

- Lower limit of subscript is ZERO whereas Upper limit is n-1.
    - n is size of array



```
int a[5];
```



```
int a[5];
```

`*(a+i)`

is equivalent to

`a[i]`

Subscript Operator

- * () is known as Subscript Operator

- It has 2 requirements i.e.
  1. On left hand side of Subscript Operator there should be a pointer (pointer constant or variable)
  2. In between the brackets of Subscript Operator there should be an integer or integral expression whose result is an integer.

Pointer Variable

Pointer Constant

```
int *ptr, a[5];

ptr = a;
```

Due to this statement address of first element will be assigned to ptr.

`ptr[1]` $\longrightarrow$ `*(ptr + 1)`

Behavior of Subscript Operator

- Using Subscript Operator we can access the elements of array
- We can do this using pointer variable or pointer constant.

```
int *ptr, a[5];

1. ptr = a;   // possible

2. a= ptr;   // not possible
```

Reason: As per assignment rule on the left side of assignment operator (=) there should be a memory location whose data type should match with RHS value i.e. variable but in line 2 a is a pointer constant not variable so it will give error.

```
a[2]; OR *(a+2);                // 1 write operation

ptr[2]; OR *(ptr+2);            // 1 write operation
                                // 1 read operation
```

```
a[2];
*(a+2);
```

- Both a and 2 are constants
- Compiler will directly go to digital address of a i.e. 452 and will add 2 according to the pointer arithmetic
- Compiler will skip 8 bytes
- And finally indirection will be performed (i.e.1 write operation)

```
ptr[2];
*(ptr+2);
```

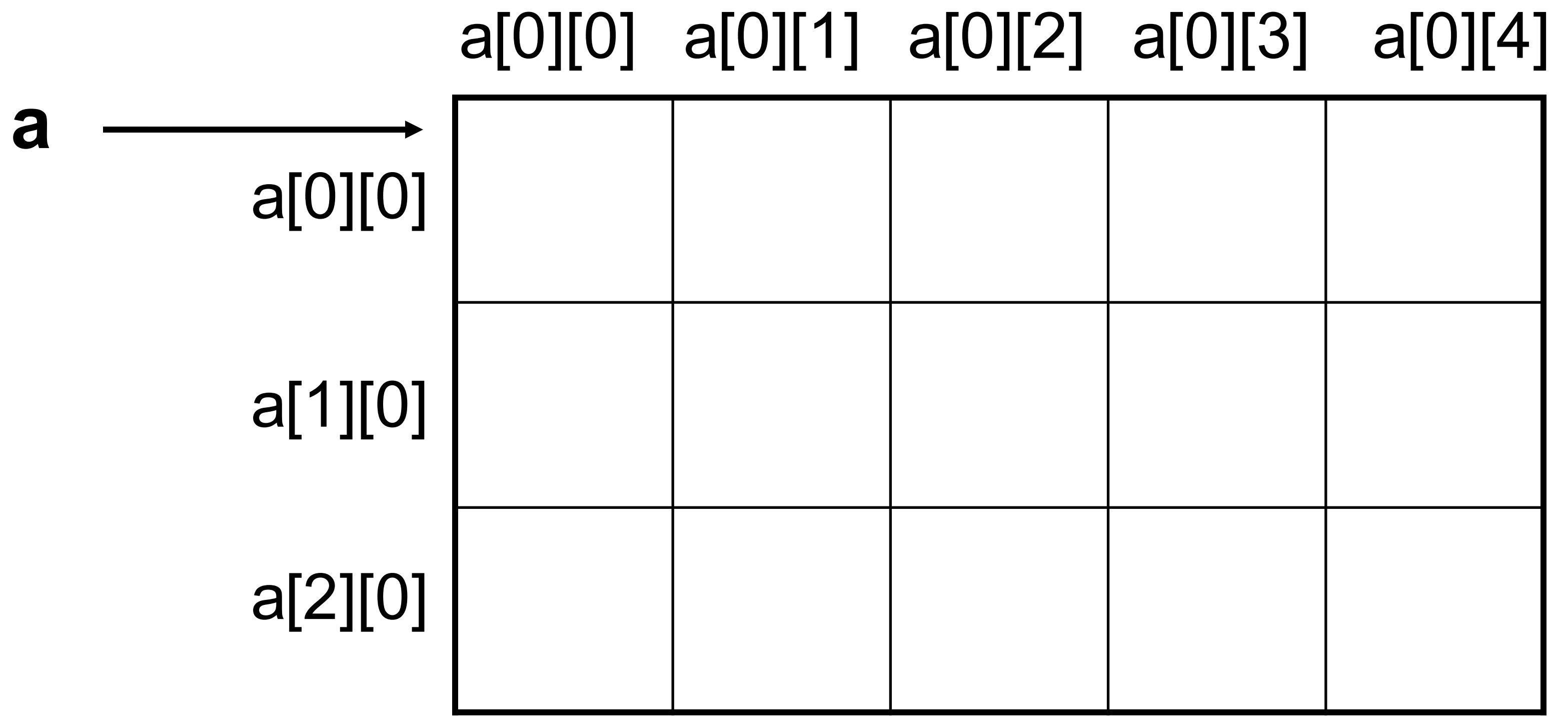
- Ptr is pointer variable and 2 is constant
- Compiler will access the ptr and will read the address stored in ptr (i.e. 1 read operation)
- Then it will add 2 according to the pointer arithmetic, and 8 bytes will be skipped.
- And finally indirection will be performed (i.e.1 write operation)

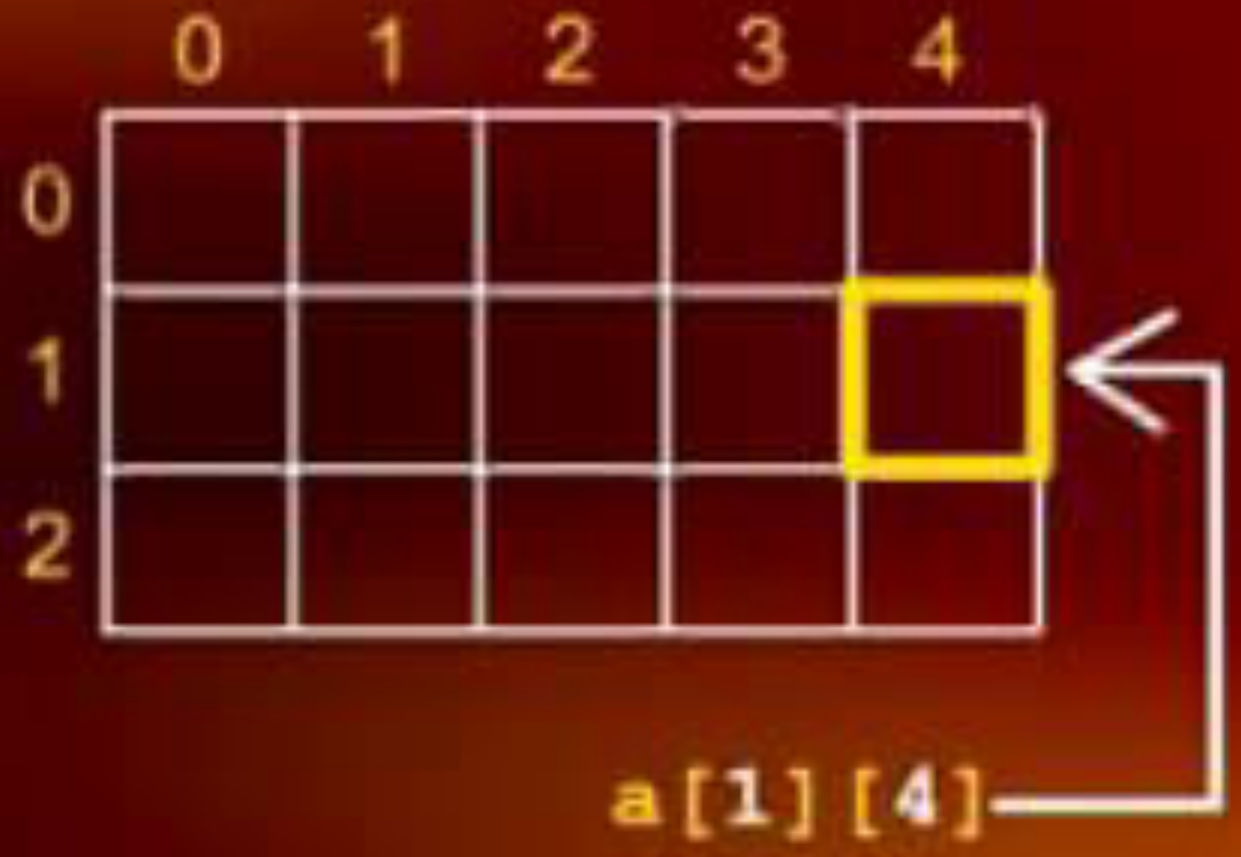
Multidimensional Arrays in C

- Multidimensional arrays are simply arrays of arrays (of arrays of arrays...)
- A multidimensional array like m[3][2] is stored in consecutive memory locations as m[0][0], m[0][1], m[1][0], m[1][1], m[2][0], m[2][1]

Two Dimensional Arrays

`a[3][5];`




0
1
2
3
4
0
1
2
a[1][4]

Two Dimensional Array

```
a[1][4]
```

OR

```
*(a+1)[4]
```

OR

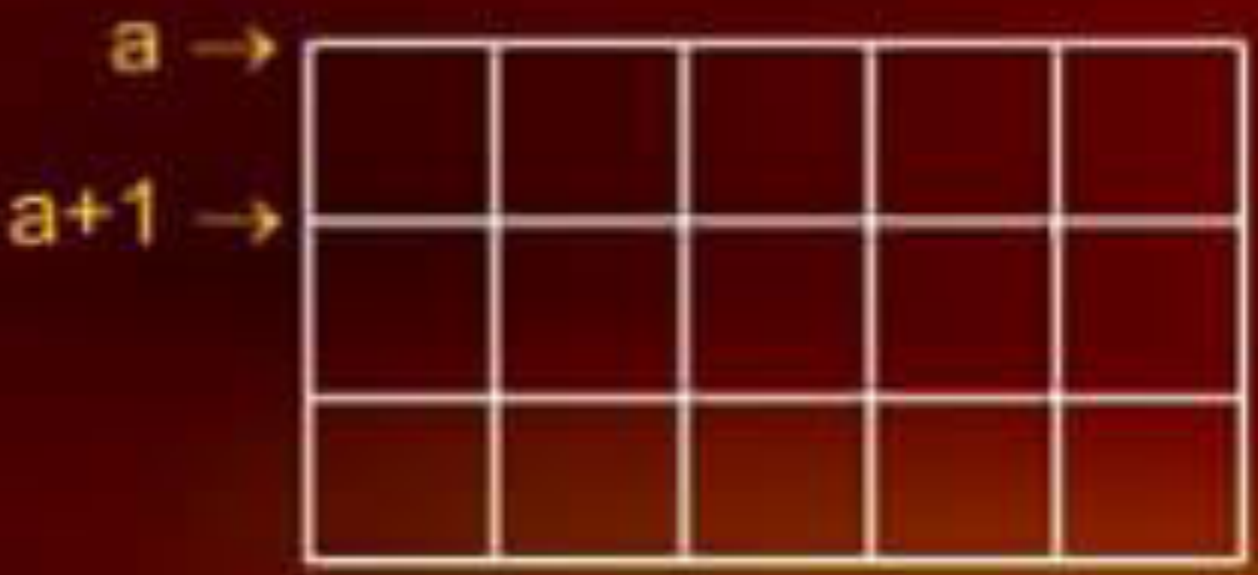
```
*(*(a+1)+4)
```

Two Dimensional Array

Name of two dimensional array is the address of its first row.

```
int a[3][5];
a + 1;    // skips 1 row(5 elements)
```

```
int a[3][5];
a+1;
```



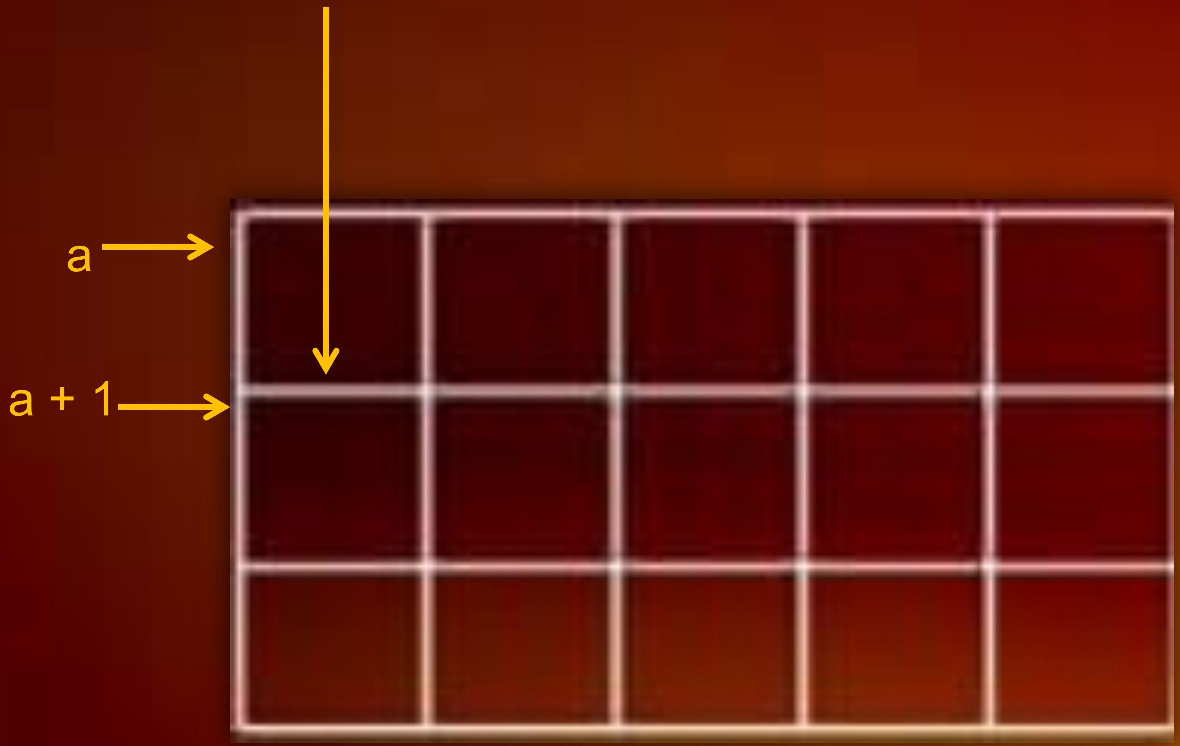
A complete row will be skipped

Two Dimensional Array

- Name of two dimensional array is the address of its first row.
- But after indirection Name of two dimensional array is the address of one element.

```
Int a [3] [5];

*(*(a+1)+4);
```

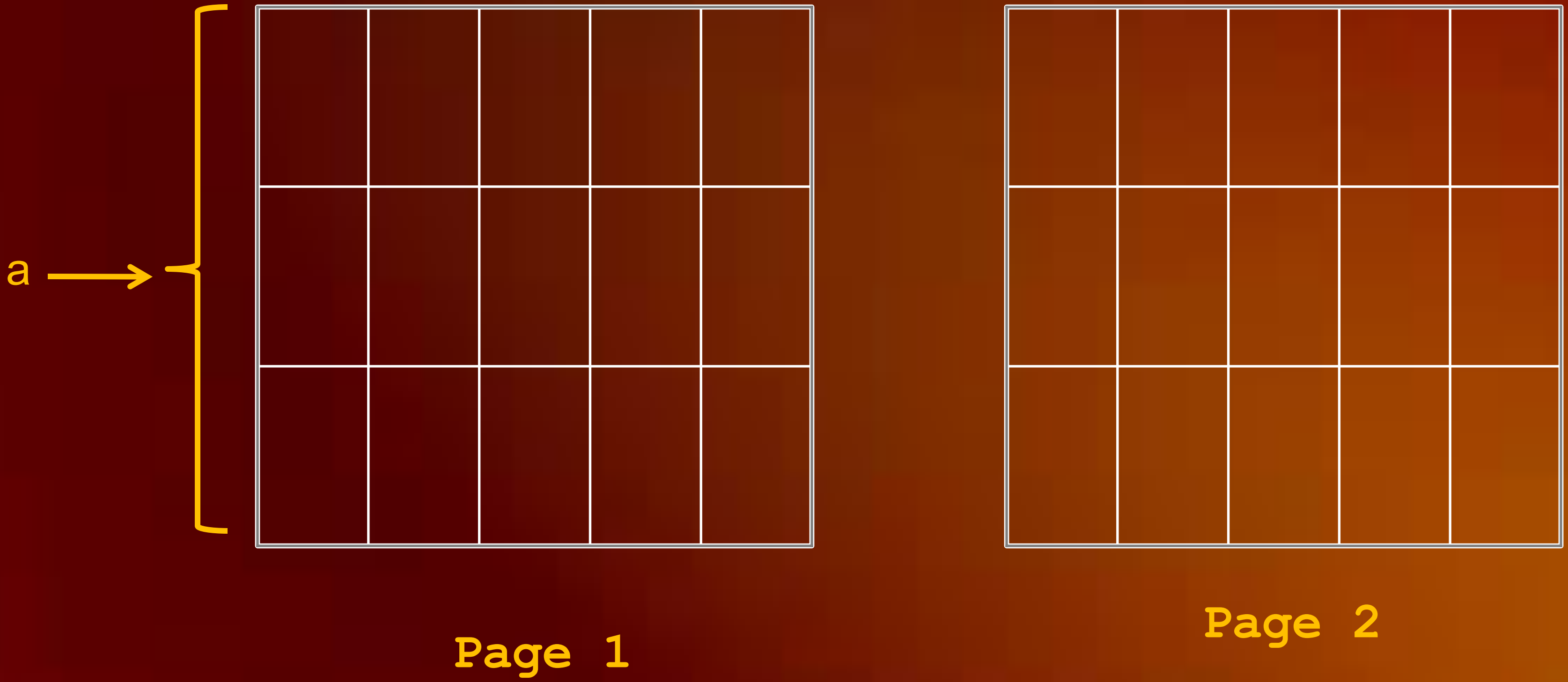


```
int (*)[5]ptr;
```

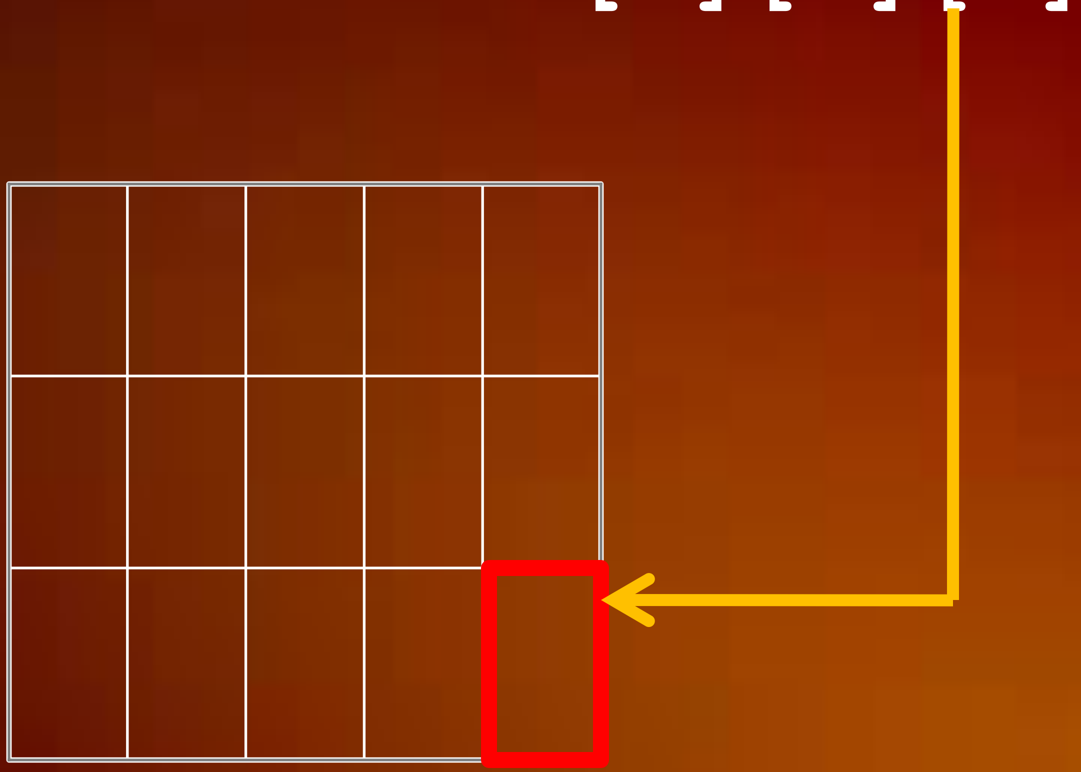
- Ptr is pointing to a row of 5 integers

```
int a[2][3][5];
```

No. of elements= 2*3*5=30



`a[1][2][4]`



Two Dimensional Array

```
int a[3][5];
```

Data type of **a** is `int (*)[5]`

OR

`int [5] *`

```
int (* ptr)[5];
ptr=a;
```

Three Dimensional Arrays

int `a[2][3][5];`

Name of three dimensional array is the address of first page in that array.

Three Dimensional Array

```
int a[2][3][5];
```

Data type of a is `int (*)[3][5];`

```
int (* ptr)[3][5];
ptr=a;
```

Three Dimensional Array

```
a[1][2][3]
```

is equivalent to

```
*(*(*(a+1)+2)+3)
```

Function Pointer

$(\;) \longrightarrow$ Function call operator

Operator Precedence

- C contains many operators, and because of operator precedence, the interactions between multiple operators can become confusing.
- Operator precedence describes the order in which C evaluates expressions

Operator Precedence

( ) operator

has higher precedence then

[ ] operator

Function Pointer

Function Pointers are pointers, i.e. variables, which point to the address of a function. You must keep in mind, that a running programme gets a certain space in the main-memory. Both, the executable compiled programme code and the used variables, are put inside this memory. Thus a function name in the program code is nothing else than an address.

Function Pointer

```
int *f1(void)
```

Function returning `int *`

```
int (*f1)(void)
```

Pointer to function returning `int`.

Function Pointer

`int **array[ ];`

array of pointers to pointers to **`int`**.

`int *( * array)( );`

pointer to function returning **`int *`**

```
double *(*fArray[10])();
```
array of pointers to functions returning `double *`

```
double *(*fArray[10])(int, int);
```
array of pointers to functions taking two `int` parameters and returning `double *`

Questions

```
double  b[2][3][4];
????????  ptr;    // data type of ptr
ptr=*b;
++ptr;                // how many bytes skipped
```

Read the declarations:

```
int ( *systemptr)(int, long *);
Char *(*(*a[50]) (void)) (void);
```