Pointers, Arrays, Multidimensional Arrays

- Pointers versus arrays
  - Lots of similarities

- How to deal with 2D, 3D, multidimensional arrays (for storing matrices and other 2D or 3D data!)

Review: Pointers

Pointers are variables that store memory addresses

Address	Memory	Name
0xeffffa94	15	a
0xeffffa98	0xeffffa94	b

```
int a = 15;
int *b = &a;
printf("%x %x %d\n", b, &b, *b);
// prints   effffa94  effffa98 15
```

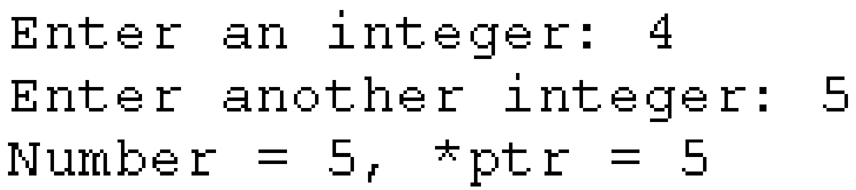
Using pointers in scanf function

```
int number;
int *ptr = &number;

printf(“Enter an integer: ”);
scanf(“%d”, &number);
printf(“Enter another integer: “);
scanf(“%d”, ptr);

printf(“Number = %d, *ptr = %d\n”, number, *ptr);
```

Read & as “at”

Don’t have to put & before ptr. It’s already an “at”.

Example output:



```
Enter an integer: 4
Enter another integer: 5
Number = 5, *ptr = 5
```

Passing pointers (addresses) to functions

```
int multiply( int *, int);

int main()
{
   int number = 3;
   int *ptr = &number;
   printf(“1: %d\n”, multiply( &number, 2 ) );
   printf(“2: %d\n”, multiply( ptr, 3 ) );
}

int multiply (int *a, int factor)
{
   return (*a) * factor;
}
```

Review: Arrays

An array is a contiguous chunk of memory to store multiple values

```
int grades[]={74,59,95,85,71,45,99,82,76};
```

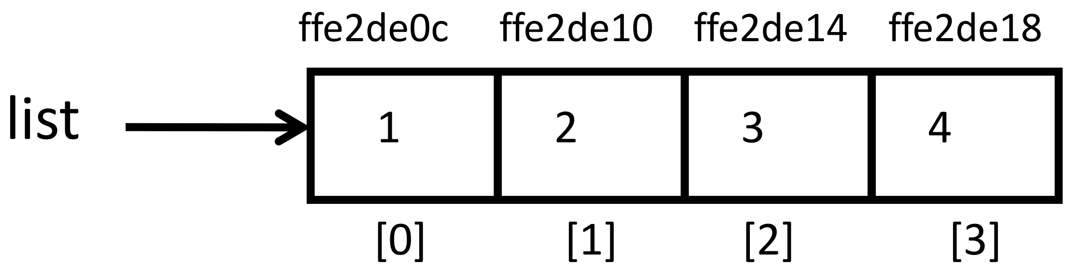
	0xeffffa00	0xeffffa04	0xeffffa08	0xeffffa0c	0xeffffa10	0xeffffa14	0xeffffa18	0xeffffa1c	0xeffffa20
grades	74	59	95	85	71	45	99	82	76
index	0	1	2	3	4	5	6	7	8

Passing arrays to functions

```
int sumArray( int [], int);

int main()
{
        int list[] = {1, 2, 3, 4};
        printf(“Sum = %d\n”, sumArray( list , 4 ));
}

int sumArray (int list[], int arraySize)
{
        int sumvalue = 0;
        for (int i=0; i<arraySize; i++)
                sumvalue += list[i];
        return sumvalue;
}
```

Array Name

The array name is a pointer to the first element of the array



list

ffe2de0c ffe2de10 ffe2de14 ffe2de18

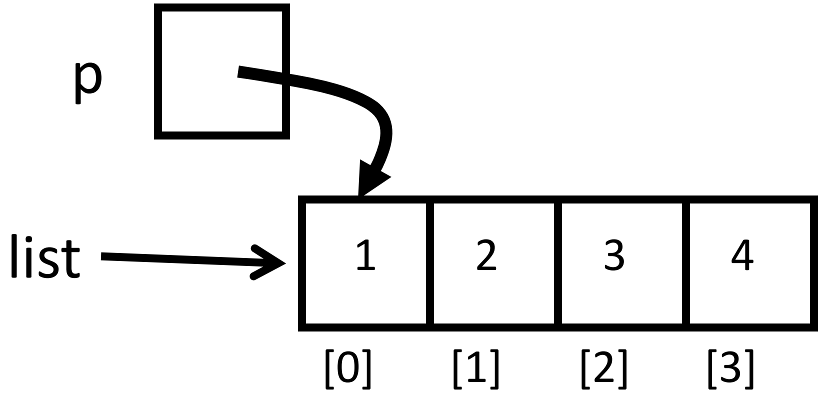
1 2 3 4

[0] [1] [2] [3]

```
int list[]={1,2,3,4};
printf("%x, %x, %d", list, &list[0], *list);
```

Output: ffe2de0c ffe2de0c 1

Pointers and Arrays

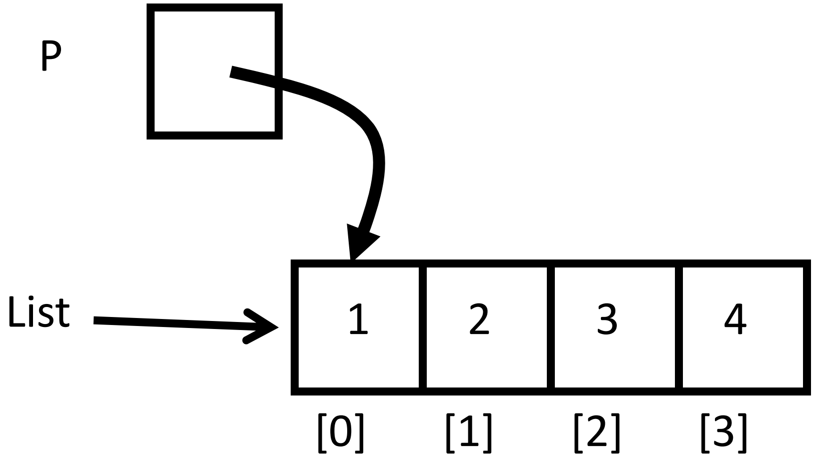


```
int *p,
int list[]={1,2,3,4};
p = list;               /* equivalent to p = &list[0] */
printf(“%d\n”, *p);   /* prints the value “1” */
```

You can use a pointer to access the array

Pointer and []

Any pointer to a block of memory can use the [] syntax, even if it is not declared as an array!



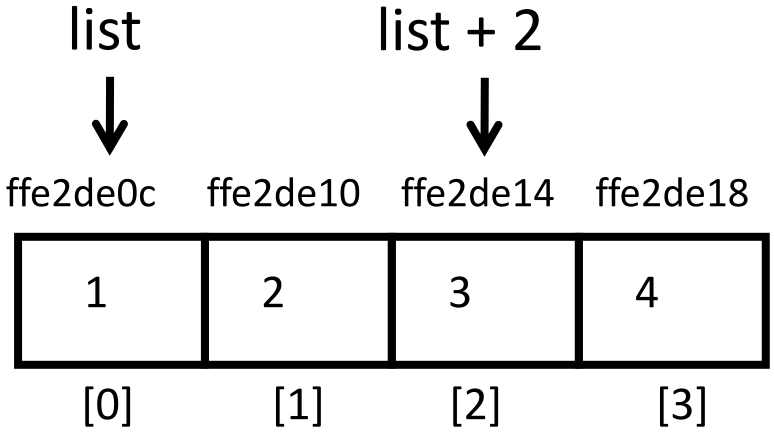
```
int *p,
int list[]={1,2,3,4};
p = list;
printf("%d\n", p[2]);     // prints 3
```

```
int *v;   and   int v[];   /* Mean the same thing */
```

Array indexing []

*list – Contents pointed to by list

*(list + 2) – Contents at list[2]



list

list + 2

ffe2de0c	ffe2de10	ffe2de14	ffe2de18
1	2	3	4
[0]	[1]	[2]	[3]

Indexing an array is just a way of finding a particular address in that block

```
int list[] = {1,2,3,4}        // array of 4 ints
       printf(“%d”, list[2]);
```

This is equivalent to

```
       printf(“%d”,*(list+2));
```

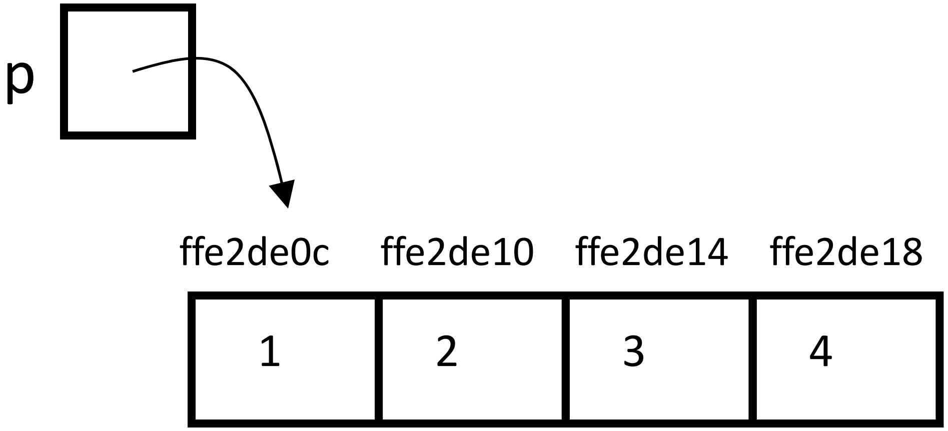
Pointer Arithmetic

When we add to a pointer, such as (p + 1), we don’t literally add 1 to the pointer address

Instead we add one “address” to the pointer

Pointer Arithmetic

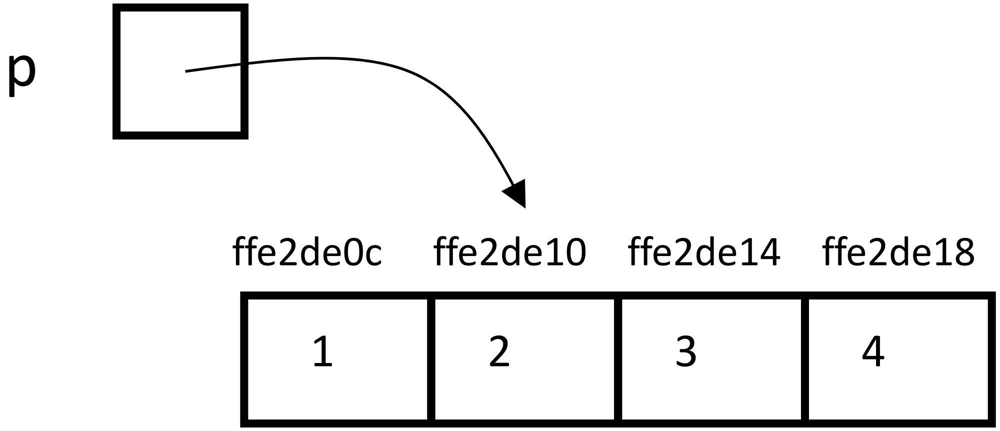
```
int list[] = {1, 2, 3, 4};
int *p = list;    /* same as p = &list[0] */
printf("%x",p);  /* prints ffe2de0c */
```



Pointer Arithmetic

```
int list[] = {1, 2, 3, 4};
int *p = list;    /* same as p = &list[0] */
printf(“%x”,p);  /* prints ffe2de0c */
p = p + 1;        /* p increases by 4 */
printf(“%x”,p);  /* prints ffe2de10 */
```

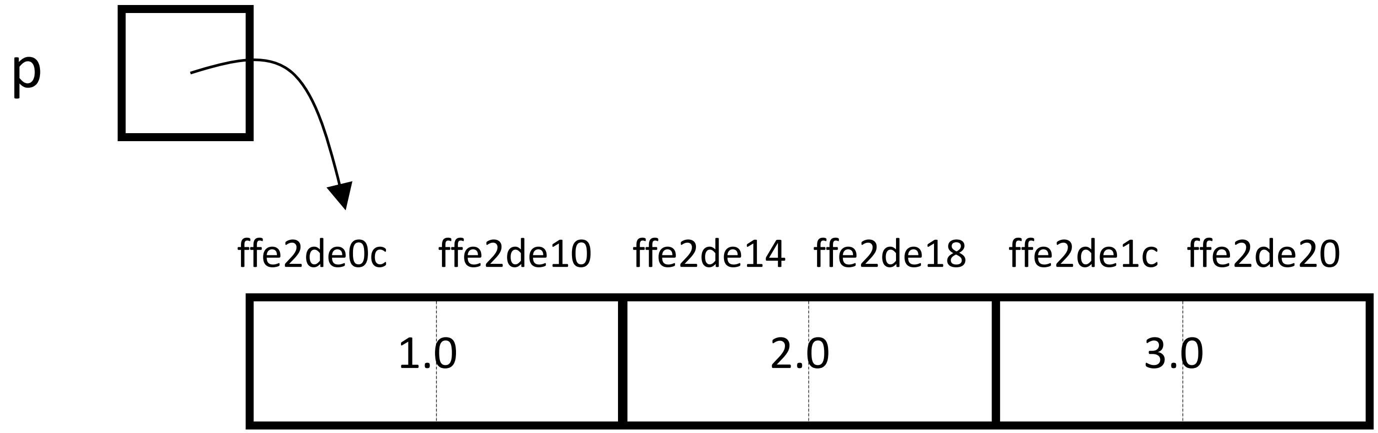
Think of pointer arithmetic as add 1 “location” instead of one byte or address.



ffe2de0c	ffe2de10	ffe2de14	ffe2de18
1	2	3	4

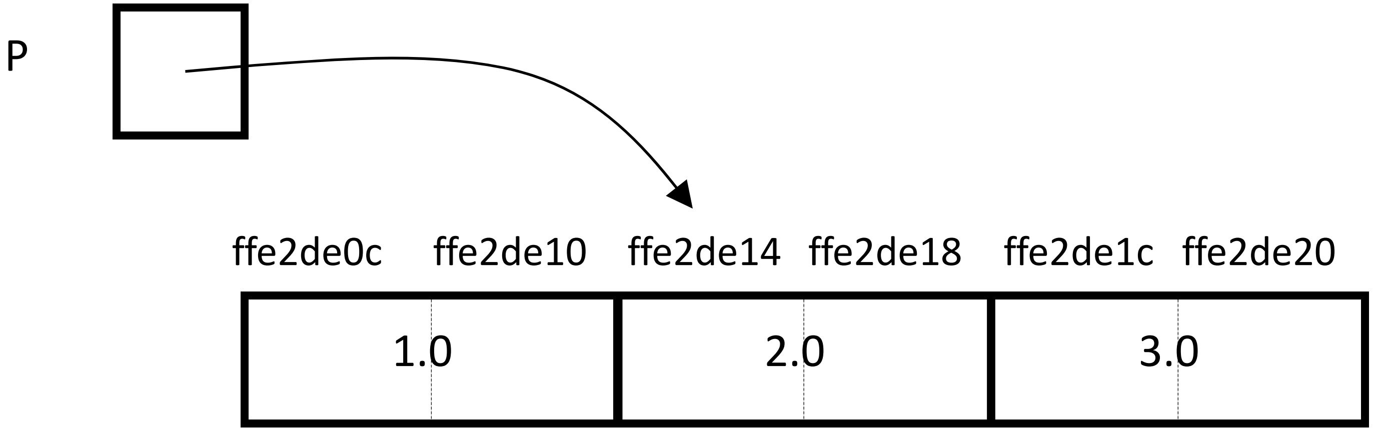
Pointer Arithmetic

```
double list2[] = {1.0, 2.0, 3.0};
double *p = list2;    /* same as p = &list2[0] */
printf(“%x”, p);      /* prints ffe2de0c */
```



Pointer Arithmetic

```
double list2[] = {1.0, 2.0, 3.0};
double *p = list2;   /* same as p = &list2[0] */
printf("%x",p);      /* prints ffe2de0c */
p = p + 1;           /* P increases by 8 bytes */
printf("%x",p);      /* prints ffe2de14 */
```



Pointer Arithmetic on Arrays

- *(list+1) references the next element in the array (equivalent to list[1])

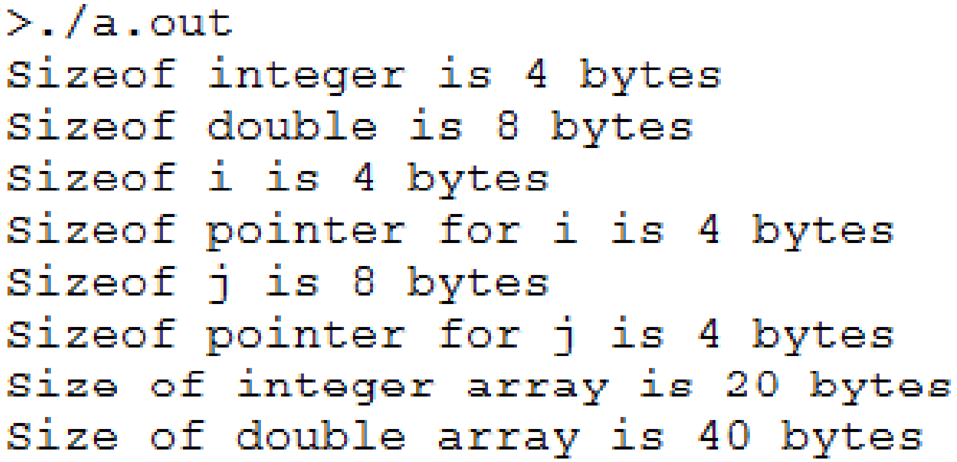
- Be careful: *(++list) works too but now we have lost our pointer to the beginning of the array!!!
  - Equivalent to: list = list + 1; *list;

sizeof() operator

Returns the number of bytes needed to store a variable or a data type

```
int i;
int *ptr4i = &i;
int IntArray[] = {1, 2, 3, 4, 5};
double j;
double *ptr4j = &j;
double doubleArray[] = {1.0, 2.0, 3.0, 4.0,5.0};

printf("Sizeof integer is %d bytes\n", sizeof(int));
printf("Sizeof double is %d bytes\n", sizeof(double));

printf("Sizeof i is %d bytes\n", sizeof(i));
printf("Sizeof pointer for i is %d bytes\n", sizeof(ptr4i));

printf("Sizeof j is %d bytes\n", sizeof(j));
printf("Sizeof pointer for j is %d bytes\n", sizeof(ptr4j));

printf("Sizeof intArray is %d bytes\n", sizeof(intArray));
printf("Sizeof doubleArray is %d bytes\n", sizeof(doubleArray));
```

sizeof() operator



```
>./a.out
Sizeof integer is 4 bytes
Sizeof double is 8 bytes
Sizeof i is 4 bytes
Sizeof pointer for i is 4 bytes
Sizeof j is 8 bytes
Sizeof pointer for j is 4 bytes
Size of integer array is 20 bytes
Size of double array is 40 bytes
```

When we pass an array

When we pass an array, we are passing the array address

```
int sumArray( int [ ], int);

int main()
{
   int list[] = {1, 2,3, 4);
   …
   sumArray( list , 4 );
   …
```

When we pass an array

This will work too (because array name is a pointer to the beginning of the array)

```
int sumArray( int *, int);
int main()
{
   int list[] = {1, 2,3, 4);
   …
   sumArray( list , 4 );
   …
```

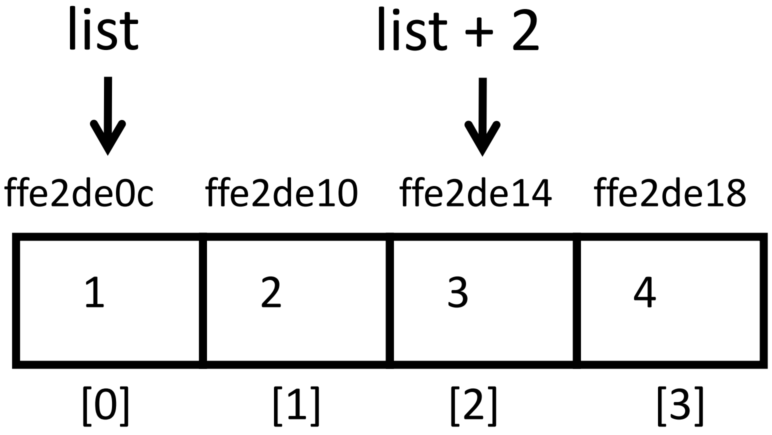
When we pass an array

- But this DOES NOT work!

```
int sumArray( int *, int);
int main()
{
   int *~~list~~ = {1, 2, 3, 4);
   …
   sumArray( list , 4 );
    …
```

Pointers

*list – Contents pointed to by list
*(list + 2) – Contents at list[2]



list

list + 2

ffe2de0c	ffe2de10	ffe2de14	ffe2de18
1	2	3	4
[0]	[1]	[2]	[3]

Indexing an array is just a way of finding a particular address in that block

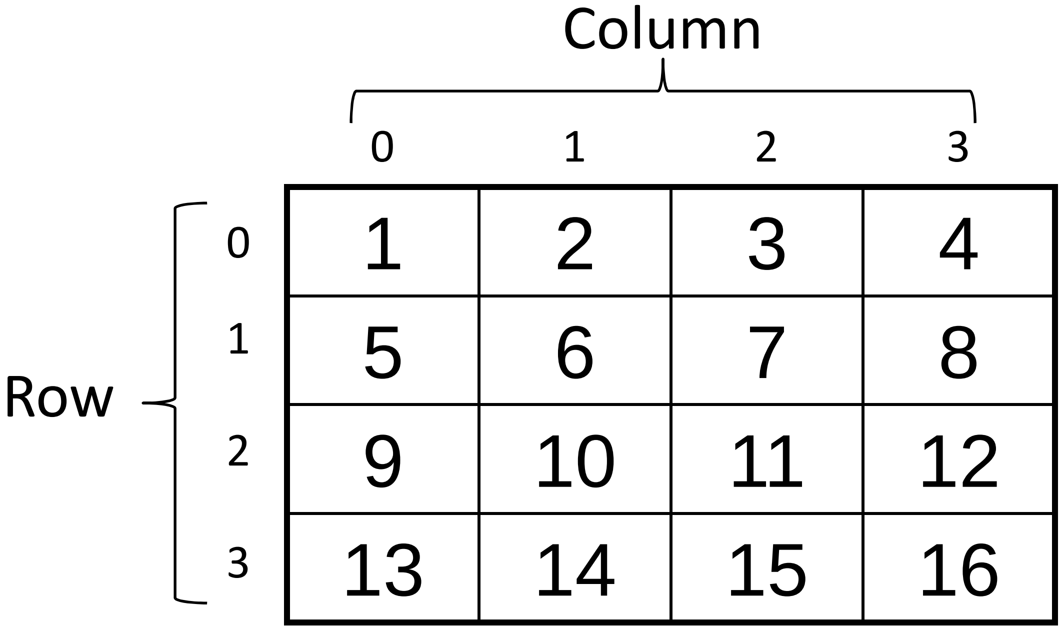
```
int list[] = {1,2,3,4}        // array of 4 ints
        printf(“%d”, list[2]);
```

This is equivalent to

```
        printf(“%d”,*(list+2));
```

2-D Arrays

```
int cave[ArraySize][ArraySize];
```

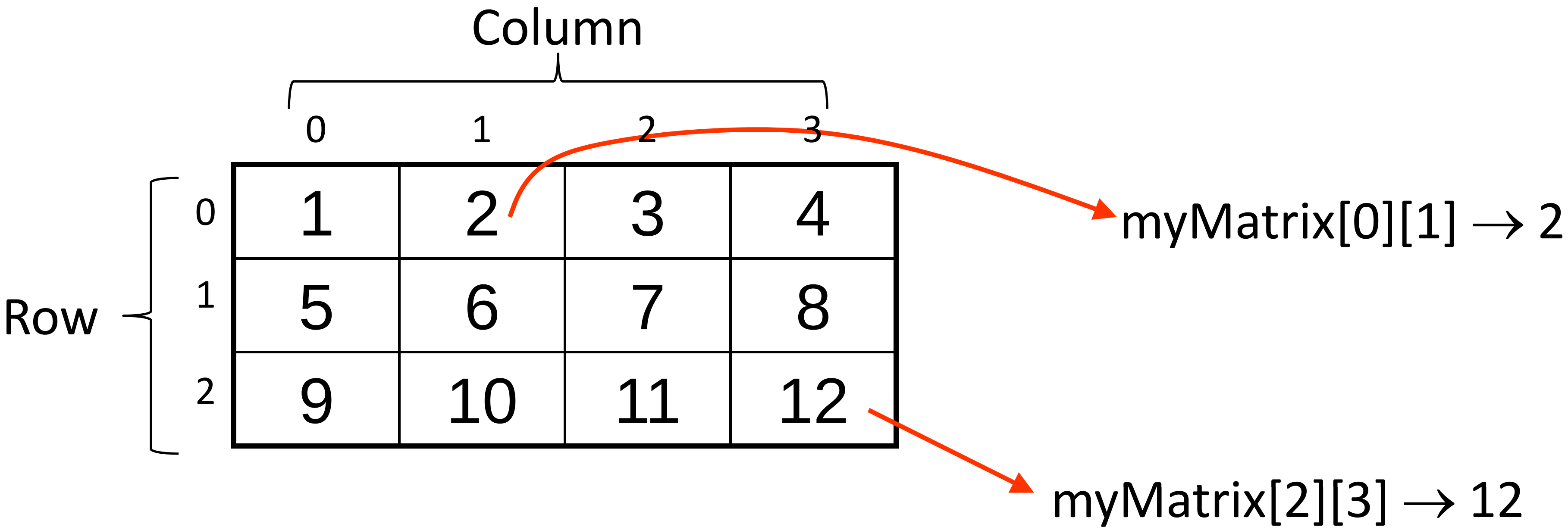


Column

Row	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

2D Arrays

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



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

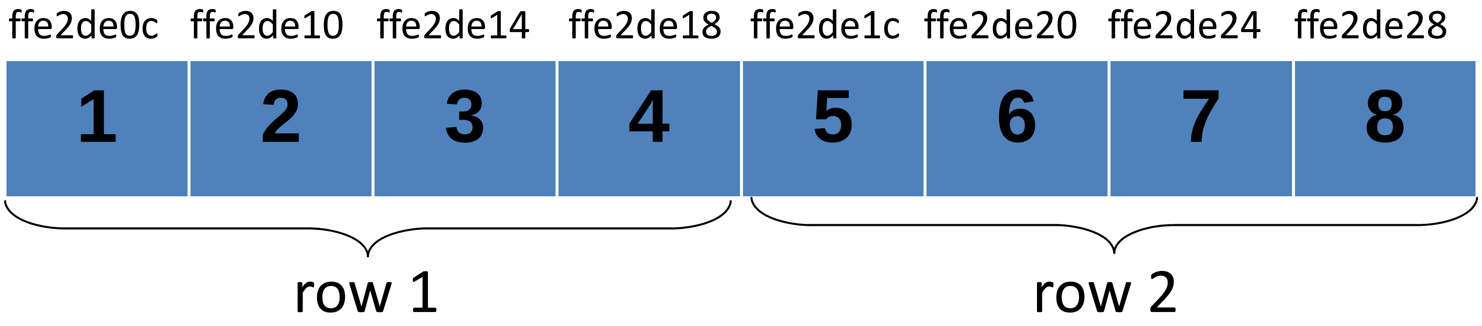
myMatrix[0][1] → 2

myMatrix[2][3] → 12

myMatrix[row][col]

Physically, in one block of memory

int myMatrix[2][4] = { {1,2,3,4},{5,6,7,8} };



ffe2de0c	ffe2de10	ffe2de14	ffe2de18	ffe2de1c	ffe2de20	ffe2de24	ffe2de28
1	2	3	4	5	6	7	8

row 1

row 2

Array elements are stored in *row major* order

Row 1 first, followed by row2, row3, and so on

2D Array Name and Addresses

int myMatrix[2][4] = { {1,2,3,4},{5,6,7,8} };

ffe2de0c	ffe2de10	ffe2de14	ffe2de18	ffe2de1c	ffe2de20	ffe2de24	ffe2de28
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

myMatrix: pointer to the first element of the 2D array
myMatrix[0]: pointer to the first row of the 2D array
myMatrix[1]: pointer to the second row of the 2D array
*myMatrix[1] is the address of element myMatrix[1][0]

Accessing 2D Array Elements

int myMatrix[2][4] = { {1,2,3,4} , {5,6,7,8} };

1	2	3	4	5	6	7	8

Indexing: myMatrix[i][j] is same as

```
*(myMatrix[i] + j)
(*(myMatrix + i))[j]
*((*(myMatrix + i)) + j)
*(&myMatrix[0][0] + 4*i + j)
```

Declaration

```
#define ROWS 3
#define COLS 5

    int table[ROWS][COLS];

    void display (table);
void display( int x[ROWS][COLS] )
{
    for (int i=0; i < ROWS; i++)
    {
          for (int j=0; j < COLS; j++ )
          {
              printf(" x[%d][%d]: %d", i, j, x[i][j]);
          }
          printf("\n");
    }
    printf("\n");
}
```

2D Arrays often require nested loops – two variables

Table A = { {13, 22, 9, 23},
{17, 5, 24, 31, 55},
{4, 19, 29, 41, 61} };

13	22	9	23	?
17	5	24	31	55
4	19	29	41	61

Table B = {1, 2, 3, 4,
5, 6, 7, 8, 9,
10, 11, 12, 13, 14 };

1	2	3	4	5
6	7	8	9	10
11	12	13	14	?

passing 2d arrays

In passing a multi-dimensional array, the first array size does not have to be specified. The second (and any subsequent) dimensions must be given!

```
int myFun(int list[][10]);
```

```
#define ROWS 3
#define COLS 5

int addMatrix( int [ ][COLS] );

int main()
{
    int a[][COLS] =  { {13, 22,  9, 23, 12}, {17,  5, 24, 31, 55}, {4, 19, 29, 41, 61}  };
    printf("Sum = %d\n", addMatrix( a ) );
}

int addMatrix( int t[ ][COLS] )
{
    int i, j, sum = 0;
    for (i=0; i<ROWS; i++)
        for (j=0; j<COLS; j++)
            sum += t[i][j];
    return sum;
}
```

1