Pointers

Pointer Operations

address = index of a byte in memory

pointer = a piece of data storing an address

T* x; declare a pointer x that will point to something of type T

&x **address of** x (get a pointer to x)

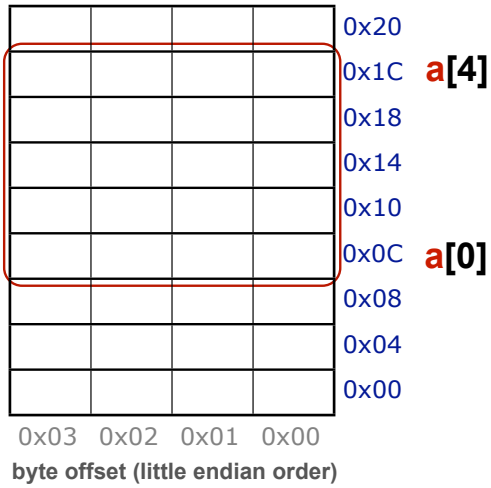
***x** **contents at** address given by pointer x (aka “dereference x” – follow the pointer)

Pointer Example

```
void do_something(int* p1, int* p2) {
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```

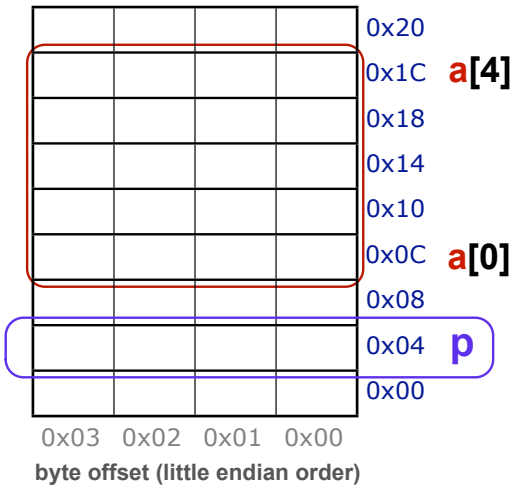

Arrays in C

```
int a[5]; // declaration

a[0] = 0xFF; // indexing
a[3] = a[0];

a[5] = 0xBAD; // uh oh
a[-1] = 0xBAD; // x2

~~a.length~~; // nope!
```



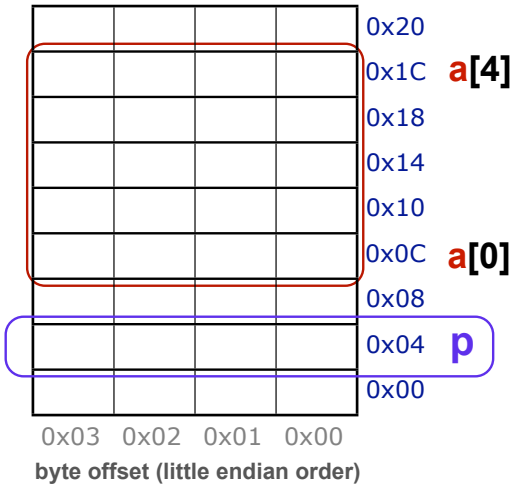
Arrays as Pointers

```
int a[5];

int* p;
```



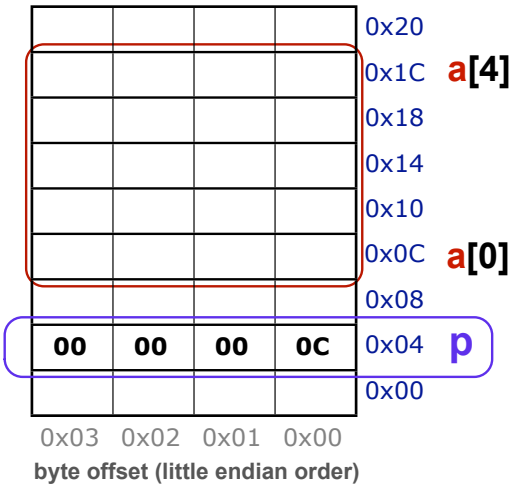
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0];
```



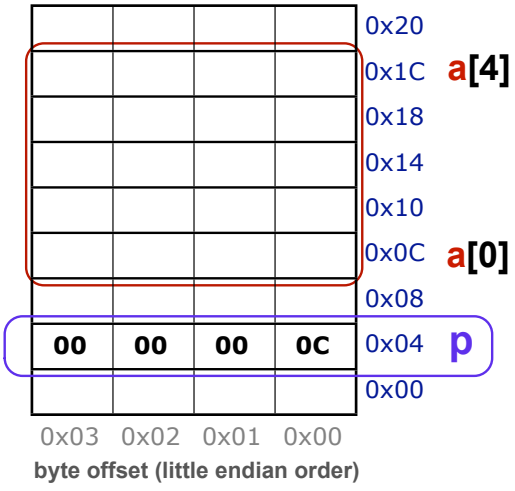
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0];
```



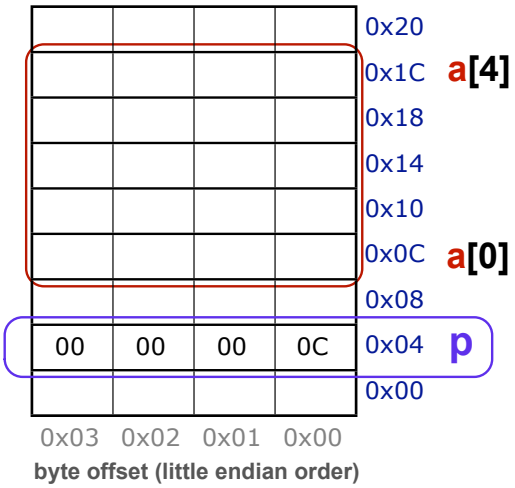
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0]; // or p = a;
```



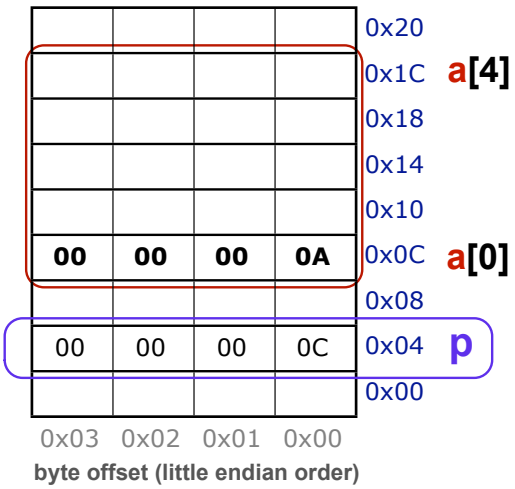
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
```



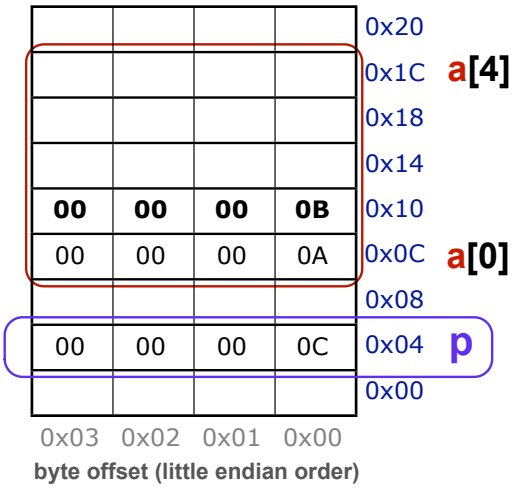
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
```



Arrays as Pointers

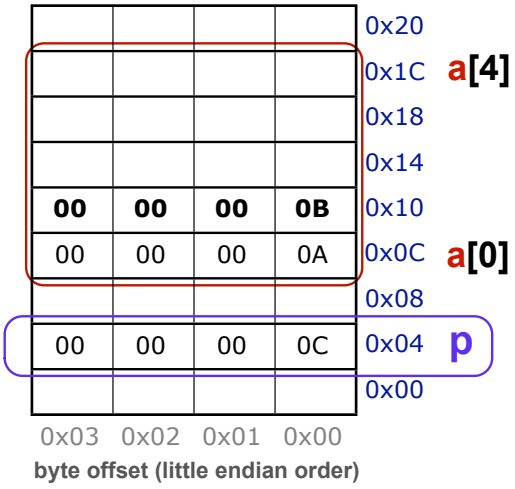
```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
p[1] = 0xB;
```



Pointer Arithmetic

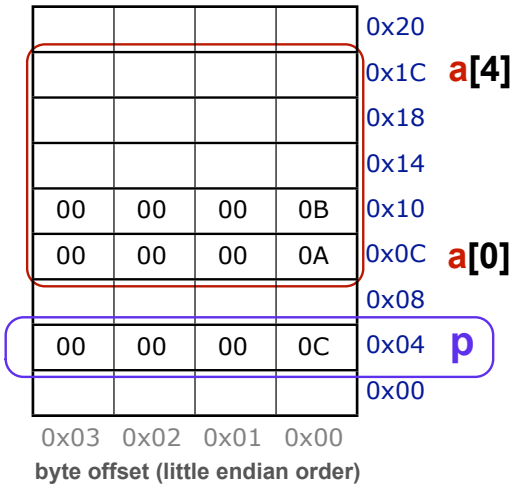
```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
p[1] = 0xB;
*(p + 1) = 0xB;
```

Equivalent!



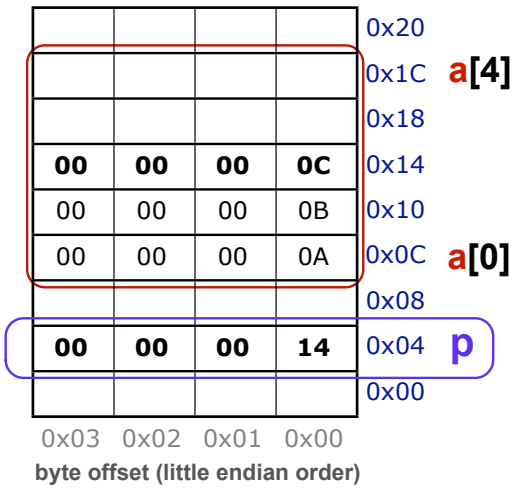
Pointer Arithmetic

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 2;
*p = a[1] + 1;
```

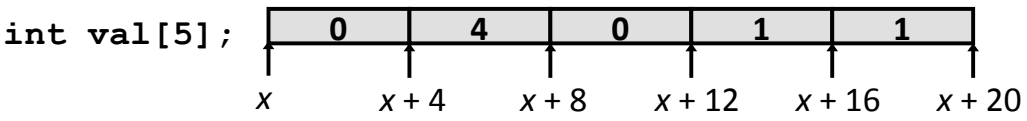


Pointer Arithmetic

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 2;
*p = a[1] + 1;
```

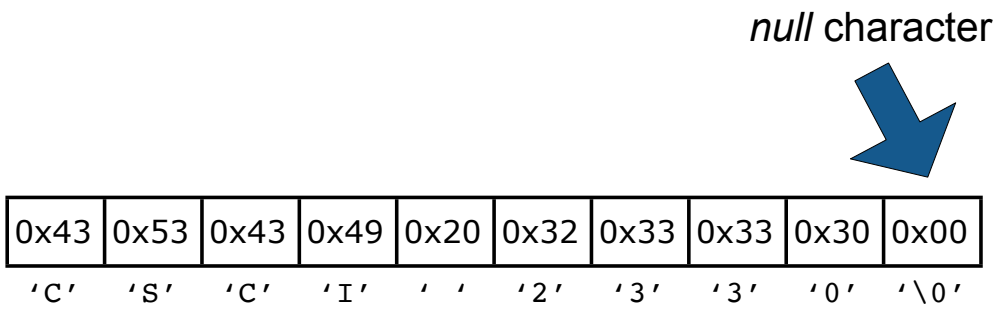


Pointer Exercises (#1)

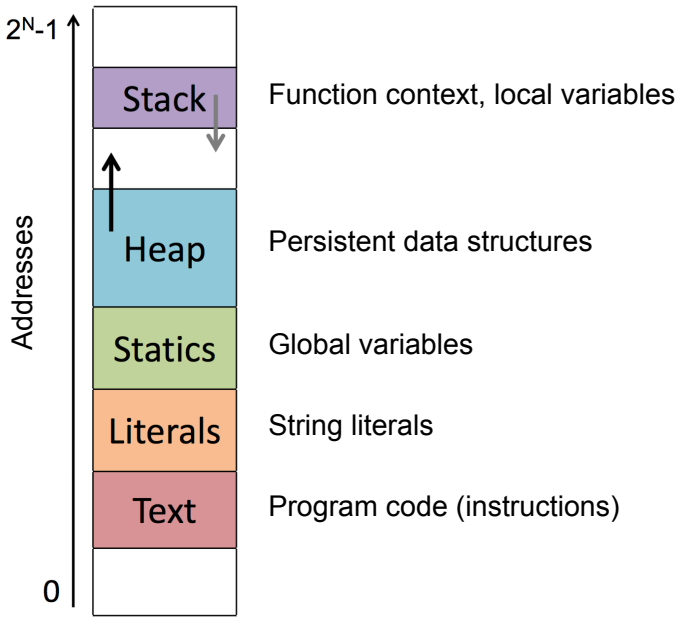


Expression	Type	Value
val[4]		
val		
val + 1		
&val[2]		
*(val + 2)		
val + i		

Null-Terminated Strings



Memory Layout



Dynamic Memory Allocation Example

```
#define ZIP_LENGTH 5

int* zip = (int*) malloc(sizeof(int) * ZIP_LENGTH);
if (zip == NULL) {            // check for error
    perror("malloc failed");  // print error message
    exit(0);                  // quit program
}

zip[0] = 0;
zip[1] = 4;
zip[2] = 0;
zip[3] = 1;
zip[4] = 1;

printf("zip is");
for (int i = 0; i < ZIP_LENGTH; i++) {
    printf(" %d", zip[i]);
}
printf("\n");

free(zip);
```