

Array and its relationship to pointer

Jinyang Li

Lesson plan

- Passing pointers to function
- Arrays
 - multiple access syntax (equivalence of pointers/arrays)
- Pointer casting

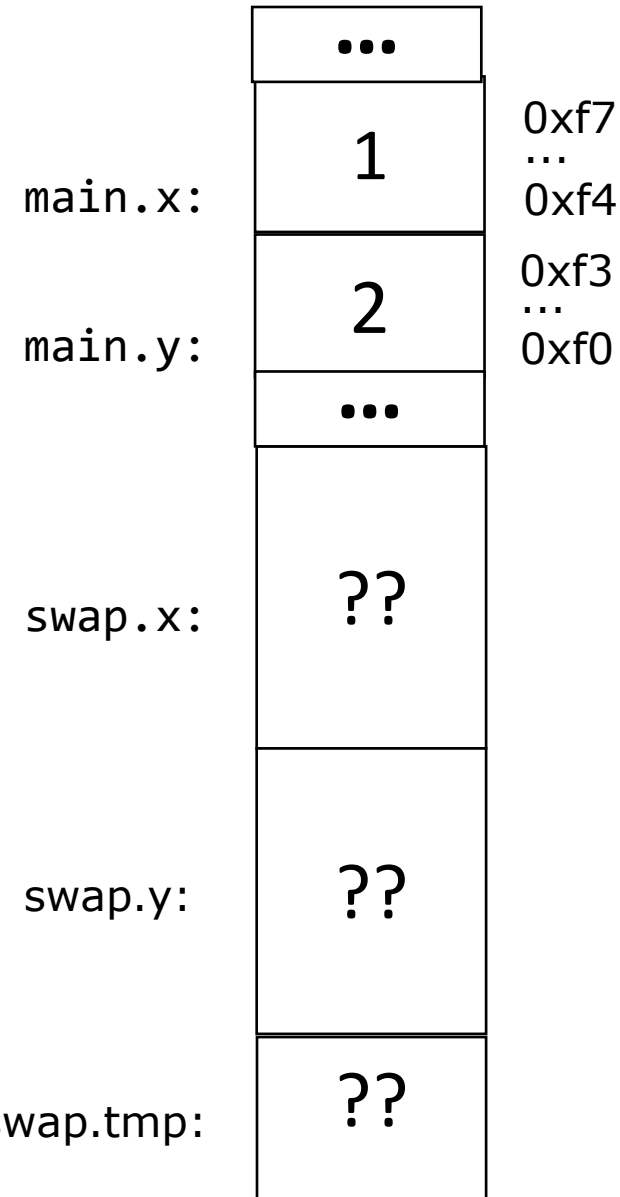
Pass pointers to function

```
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

    printf("x:%d, y:%d", x, y);
}
```

Size and value of x, y, tmp
in swap upon function entrance?

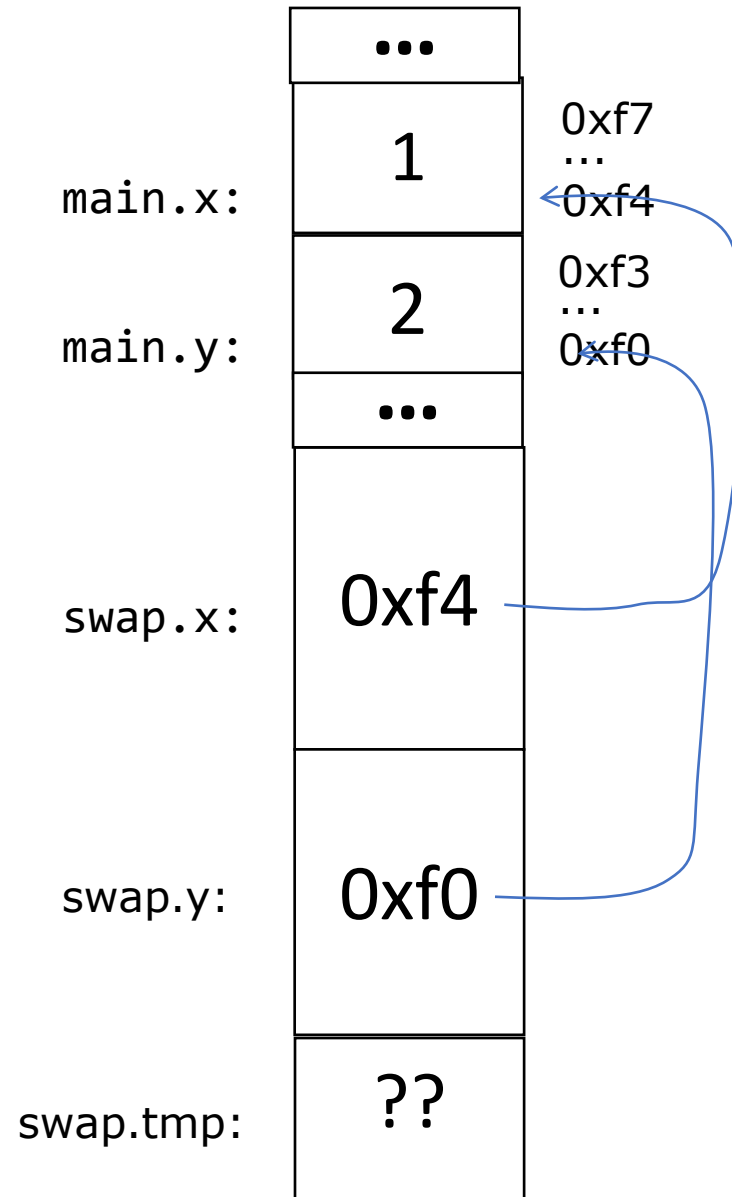


Pass pointers to function

```
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

    printf("x:%d, y:%d", x, y);
}
```

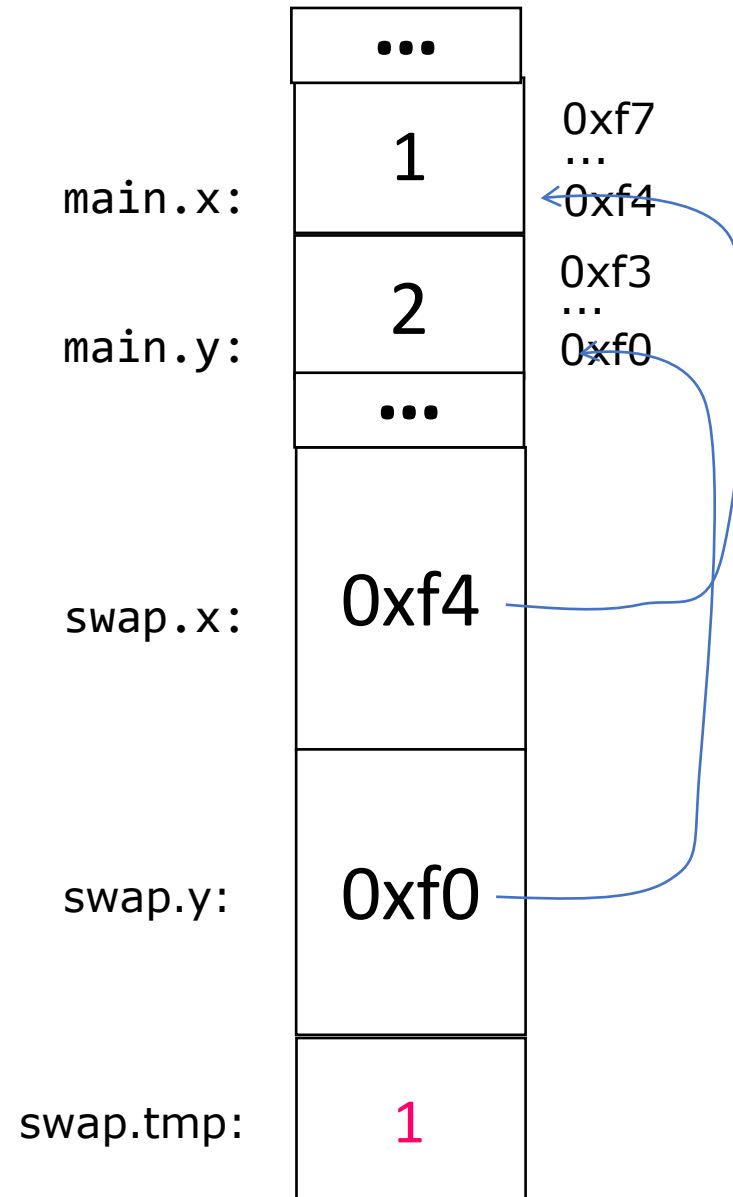


Pass pointers to function

```
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

    printf("x:%d, y:%d", x, y);
}
```

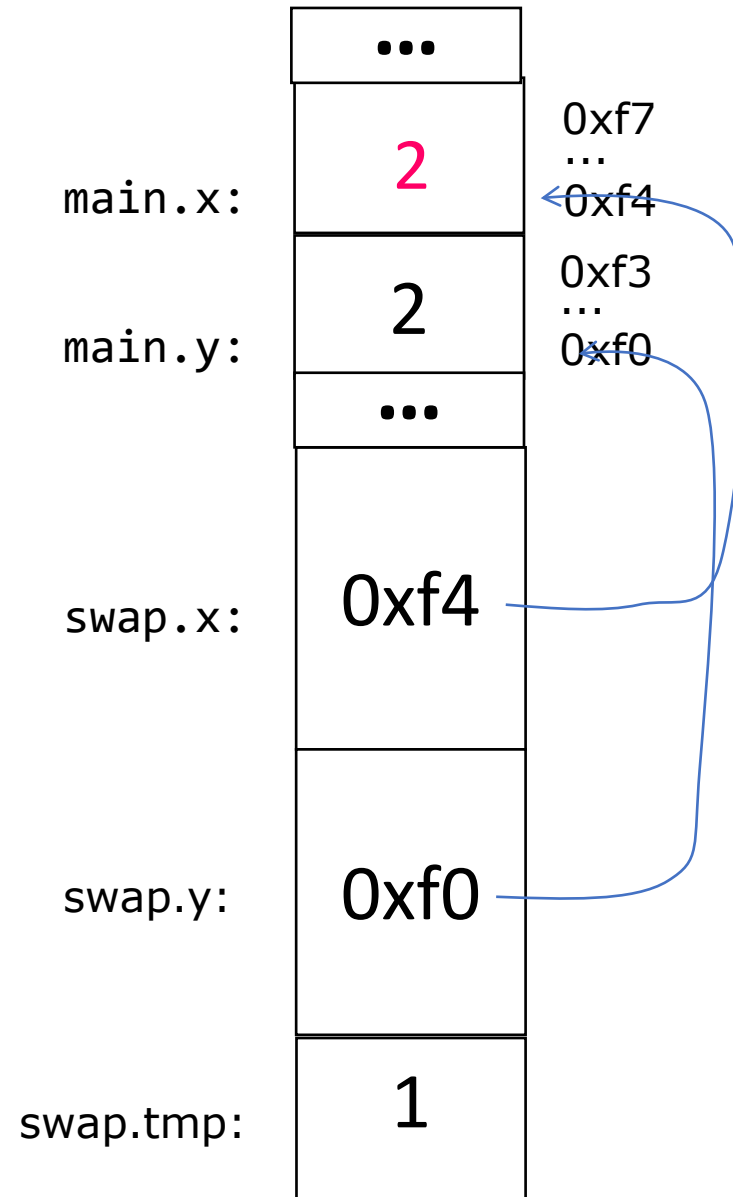


Pass pointers to function

```
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

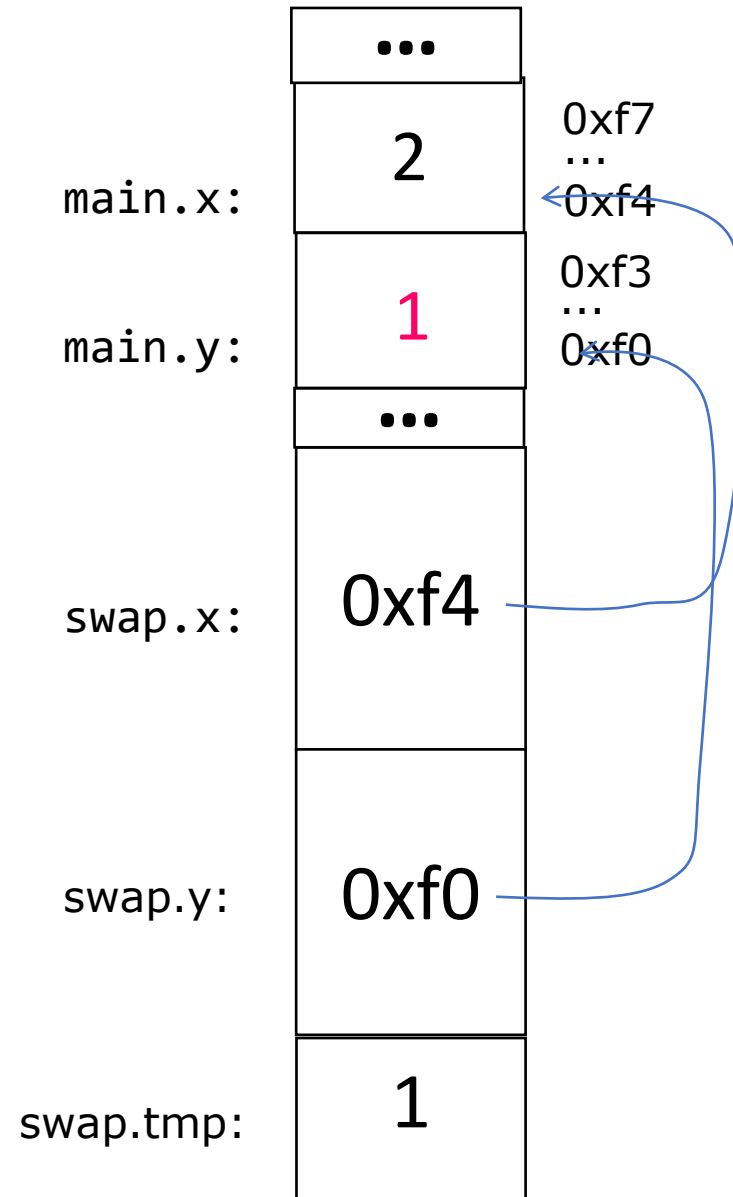
    printf("x:%d, y:%d", x, y);
}
```



Pass pointers to function

```
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

    printf("x:%d, y:%d", x, y);
}
```

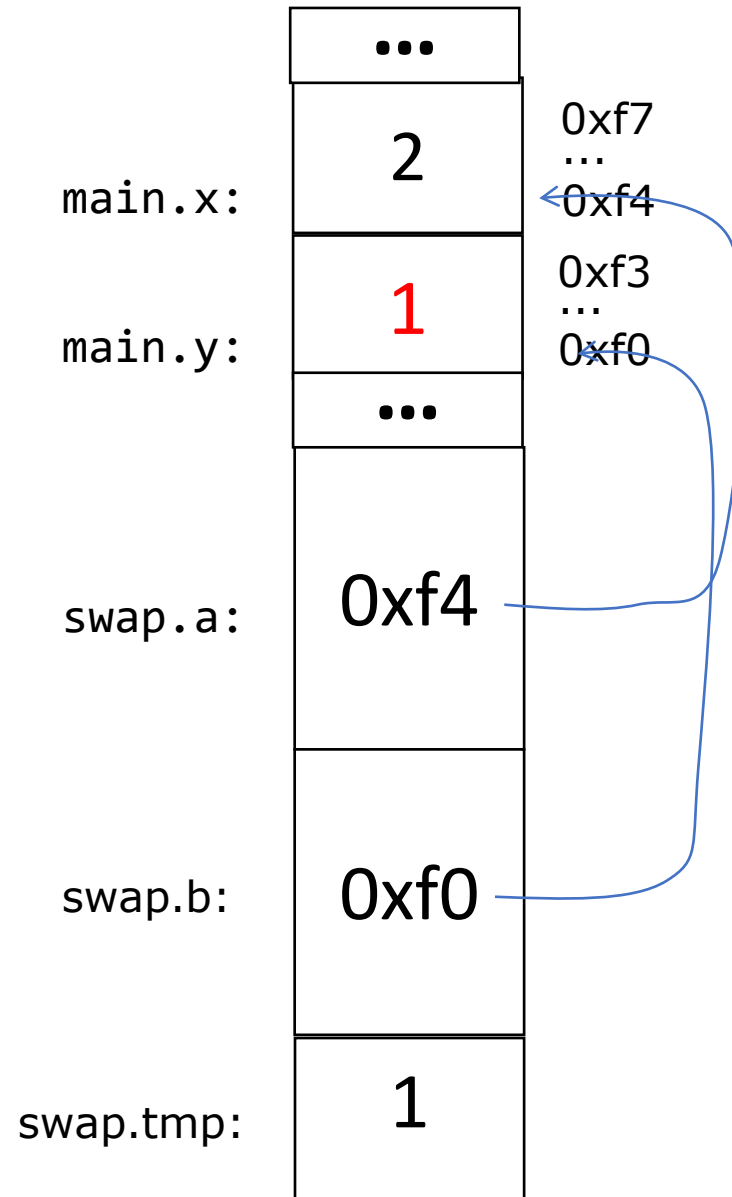



Pass pointers to function

```
void swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y);

    printf("x:%d, y:%d", x, y);
}
```



Arrays

Array is a collection of contiguous objects with
the same type

Array

a[3]:	0	0x11c
a[2]:	3	0x118
a[1]:	2	0x114
a[0]:	1	0x110
a:		...
		...
		...
		...

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

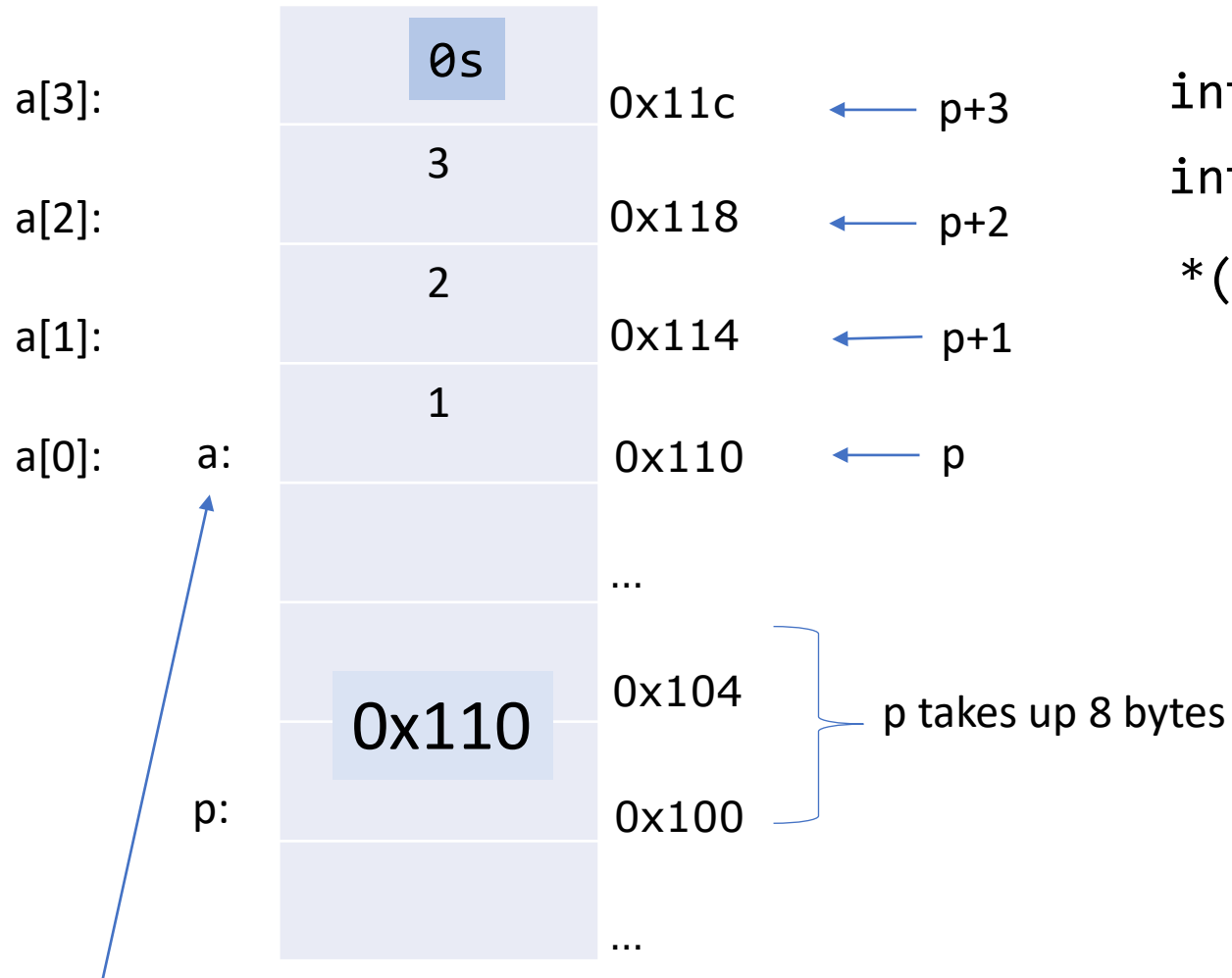
```
a[3] = 0;
```

Method-1: access using index



There's no meta-data (e.g. capacity, length) associated/stored with the array

Array access using pointer



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

```
int *p = a;
```

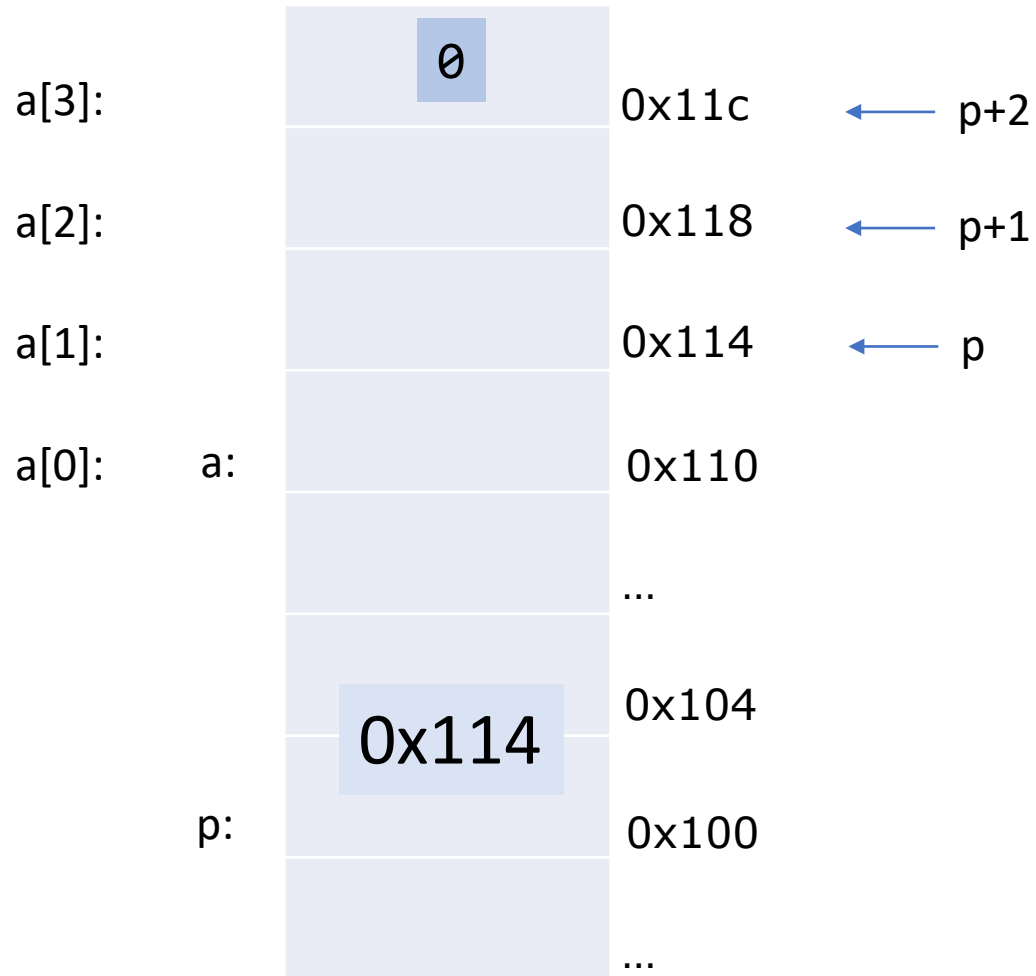
```
*(p+3)=0; Method-2: access using pointer (arithmetic)
```

- `p+i` points to the `i`-th element after the one pointed to by `p`
 - i.e. `p+i` is calculated as: `p's value + sizeof(*p) * i`
- `p-i` points to the `i`-th element before the one pointed to by `p`

Built-in function `sizeof` returns size (in bytes) of the object representing a given expression or type

`a` (array name) is aliased to be the memory address of the first element.
`a` is effectively a constant, not a variable, cannot be changed

Array access using pointer

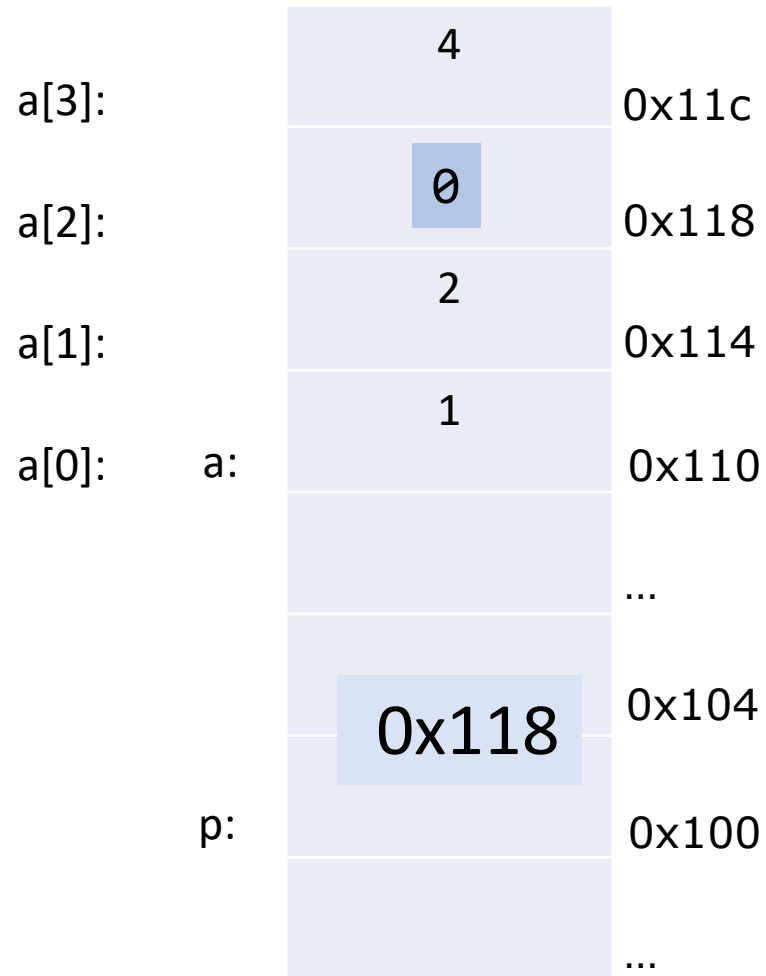


```
int a[4];  
int *p = &a[1];  
*(p+2)=0;
```

`&a[i]` is syntactically equivalent to: `a+i`

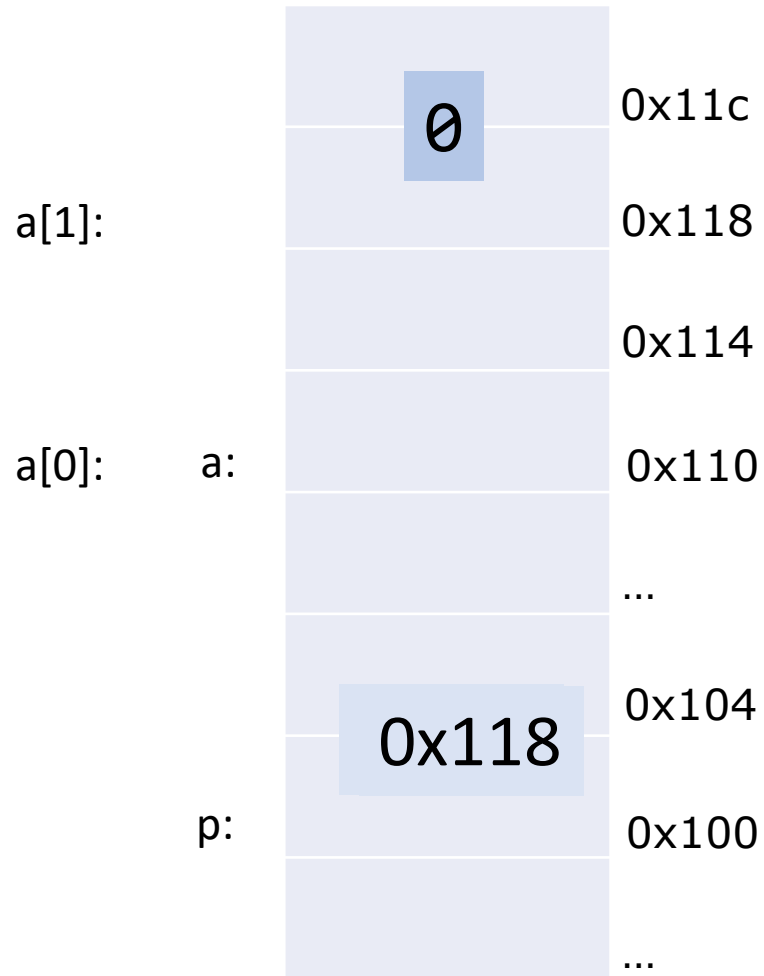
`*(p+i)` is syntactically equivalent to: `p[i]`

Array access using pointer



```
int a[4] = {1, 2, 3, 4};  
int *p = &a[3];  
  
p--;  
  
*p=0;
```

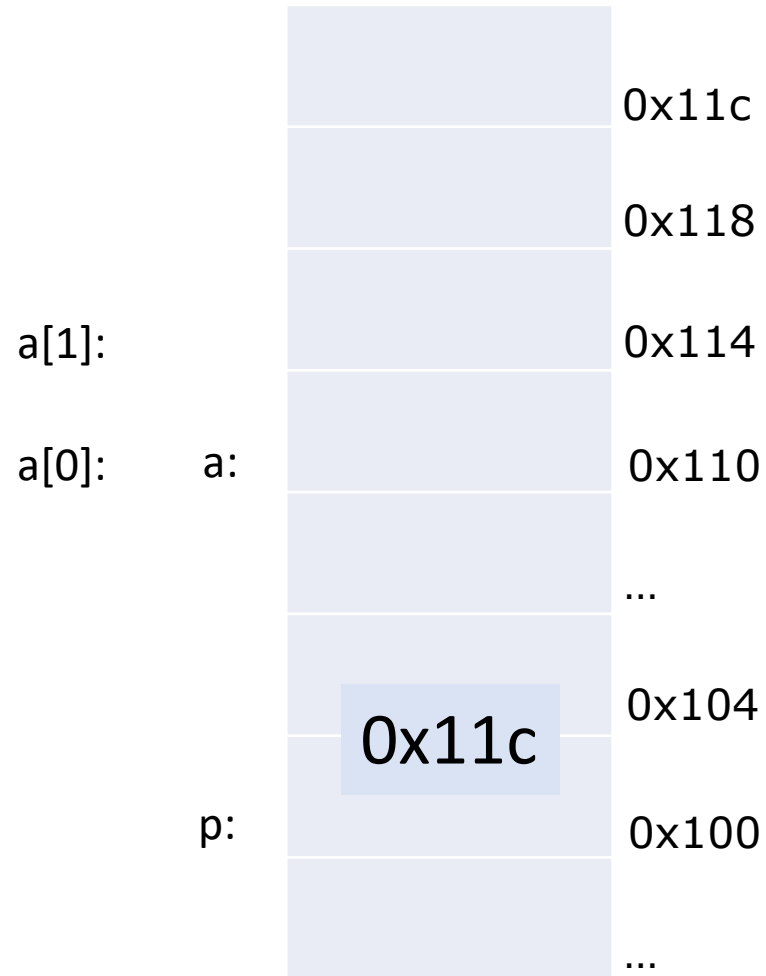
Array access using pointer



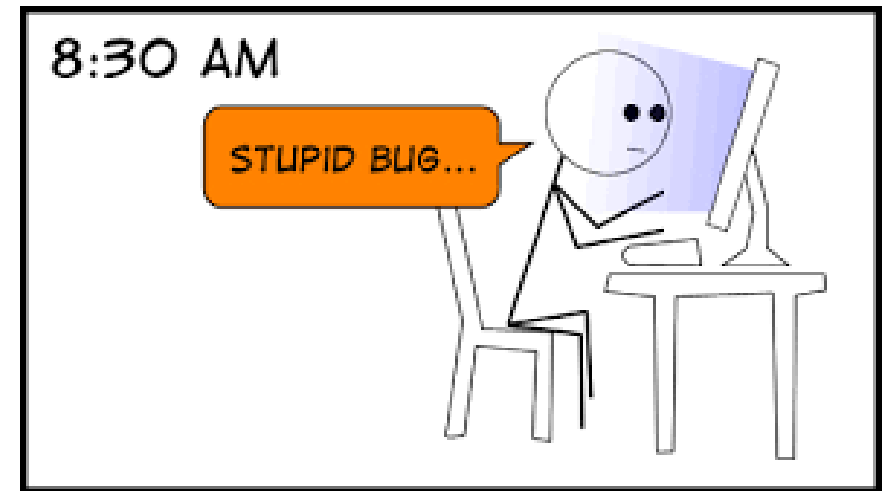
```
char *a[2];  
char ** p = &a[0];  
  
p++;  
  
*p=NULL;
```

Equivalent to:
`p = a`

Out-of-bound access results in (potentially silent) memory error



```
int a[2];  
int *p = a;  
  
p += 3;  
  
*p=0;
```





Breakout time!

Pass array to function via pointer

```
// multiply every array element by 2
void multiply2(int *a) {
    for (int i = 0; i < ???; i++) {
        a[i] *= 2;
    }
}
```

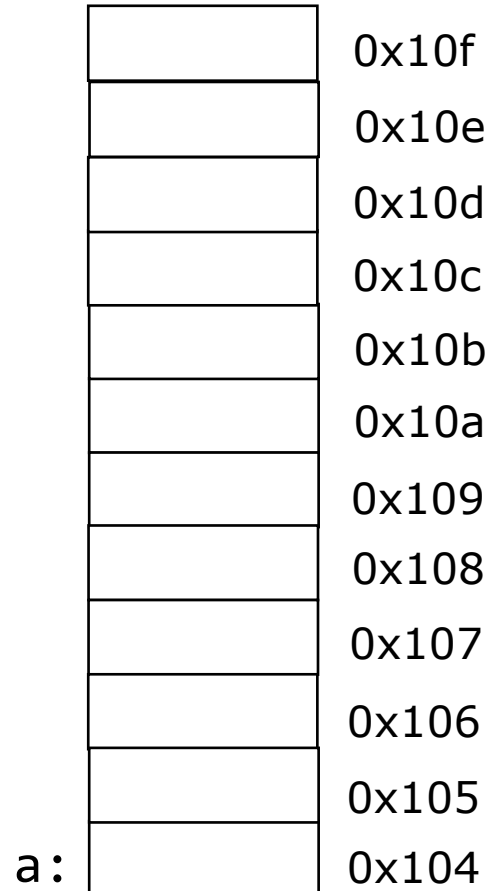
```
int main() {
    int a[2] = {1, 2};
    multiply2(a);
    for (int i = 0; i < 2; i++) {
        printf("a[%d]=%d", i, a[i]);
    }
}
```

Pass array to function via pointer

```
// multiply every array element by 2
void multiply2(int *a, int n) {
    for (int i = 0; i < n; i++) {
        a[i] *= 2; // (*(a+i)) *= 2;
    }
}

int main() {
    int a[2] = {1, 2};
    multiply2(a, 2);
    for (int i = 0; i < 2; i++) {
        printf("a[%d]=%d", i, a[i]);
    }
}
```

Pointer casting



```
int a = 0x12345678;  
char *cp = (char *)&a;
```

```
// What is *cp?
```

```
cp++;
```

```
// What is *cp?
```

Assume 64-bit small endian machine

Another example use of pointer casting

```
unsigned int extract_float_bit_pattern(float f)
{
    unsigned int i = *(unsigned int *)&f;
    return i;
}
```

Summary

- Arrays:
 - No array meta-data associated/stored. No bound checking
 - equivalence of pointer arithmetic and array access
 - `p+i` same as `&p[i]`
 - `*(p+i)` same as `p[i]`
 - Value of `p+i` is computed as `p+sizeof(*p)*i`
- Pass pointers to functions
- Pointer casting