CS 33

Introduction to C
Part 2

Swapping

Write a function to swap two ints

```
void swap(int i, int j) {
  int tmp;
  tmp = j; j = i; i = tmp;
}
int main() {
   int a = 4;
   int b = 8;
   swap(a, b);
   printf("a:%d  b:%d", a, b);
}
```

```
$ ./a.out
a:4  b:8
```

Why “pass by value”?

- Fortran, for example, passes parameters “by reference”
- Early implementations had the following problem (shown with C syntax):

```
int main() {
   function(2);
   printf("%d\n", 2);
}
void function(int x) {
   x = 3;
}
```

```
$ ./a.out
3
```

Variables and Memory

What does

```
int x;
```

do?

- **It tells the compiler:**

  I want *x* to be the name of an area of memory that's big enough to hold an *int*.

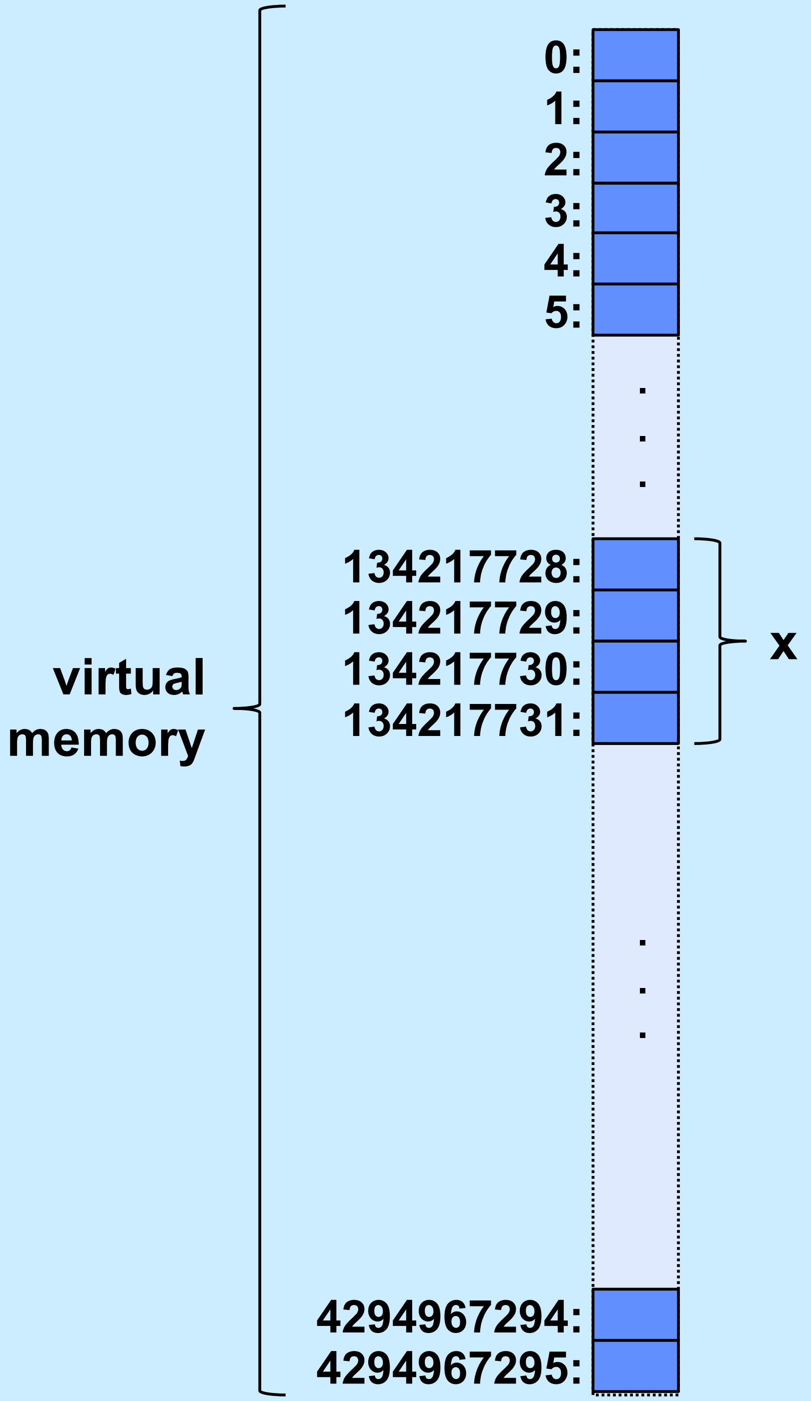
What's memory?

Memory

- “Real” memory
  - it’s complicated: we discuss it later in the course
  - it involves electronics, semiconductors, physics, etc.
  - it’s not terribly relevant at this point
- “Virtual” memory
  - the notion of memory as used by programs
  - it involves logical concepts
  - it’s how you should think about memory (most of the time)

Virtual Memory

- It's a large array of bytes
  - one byte is eight bits
  - an int is four consecutive bytes
  - so is a float
  - a char is one byte
- The array index of a byte is its *address*
  - the address of a larger item is the index of its first byte

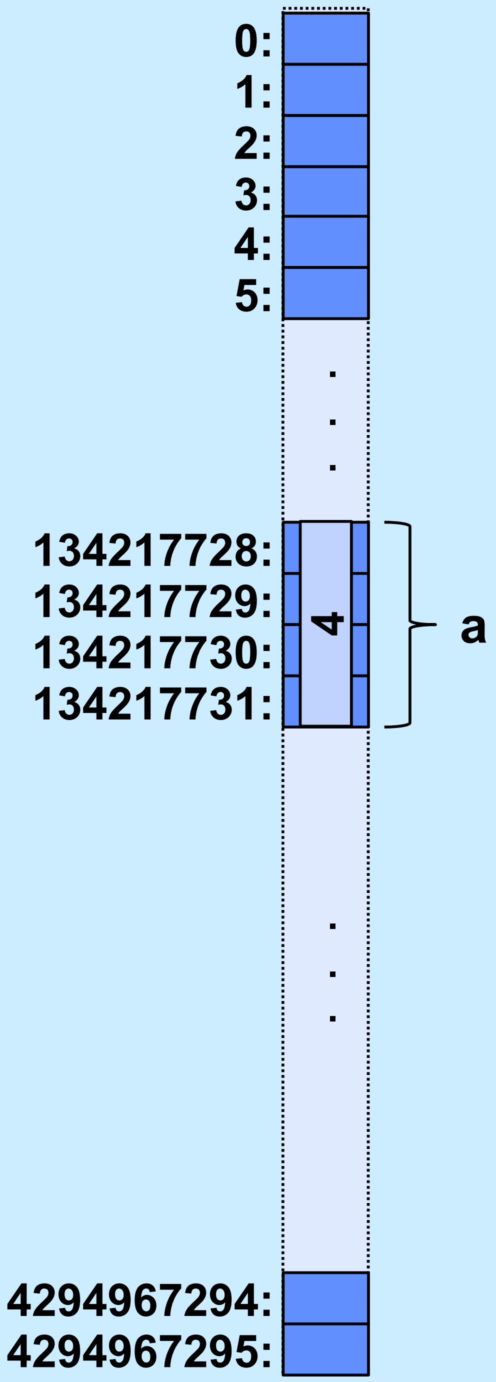


Memory addresses in C

- In C
  - you can get the memory address of any variable
  - just use the operator &

```
int main() {
    int a = 4;
    printf("%u\n", &a);
}
```

```
$ ./a.out
134217728
```



C Pointers

- What is a C pointer?
  - a variable that holds an address
- Pointers in C are “typed” (remember the promises)
  - pointer to an int
  - pointer to a char
  - pointer to a float
  - pointer to <whatever you can define>
- C has a syntax to declare pointer types
  - things start to get complicated …

C Pointers

p is a pointer to an int

if you follow p, you find an int

```c
int main() {
    int *p;
    int a = 4;
    p = &a;
    printf("%u\n", p);
}
```

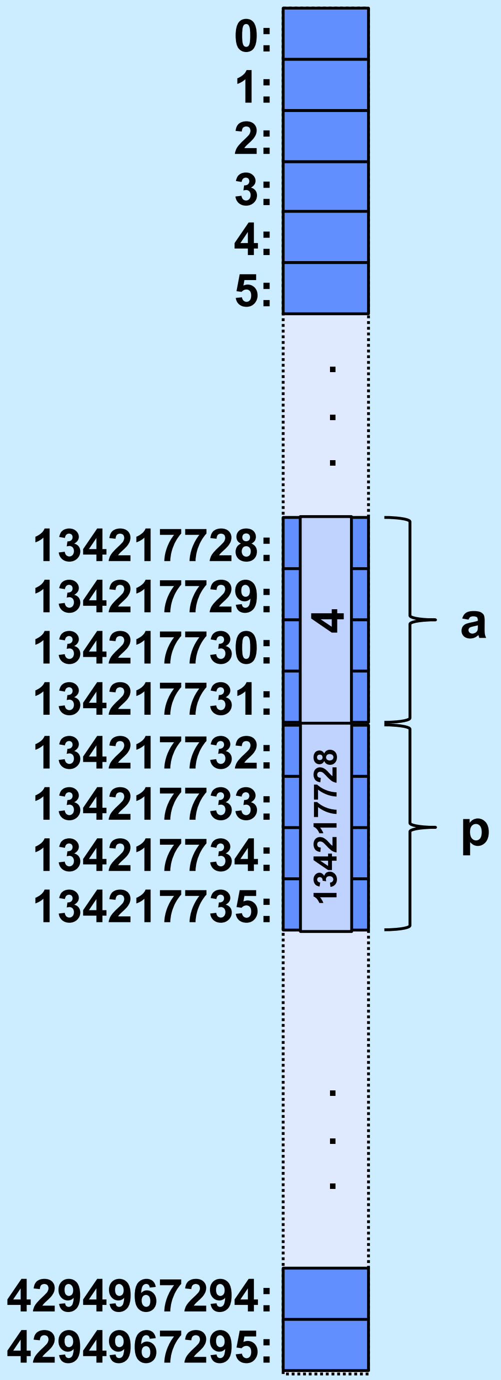
p is assigned the address of a

```
$ ./a.out
134217728
```

C Pointers

```
int main() {
    int *p;
    int a = 4;
    p = &a;
    printf("%u\n", p);
}
```

```
$ ./a.out
134217728
```



C Pointers

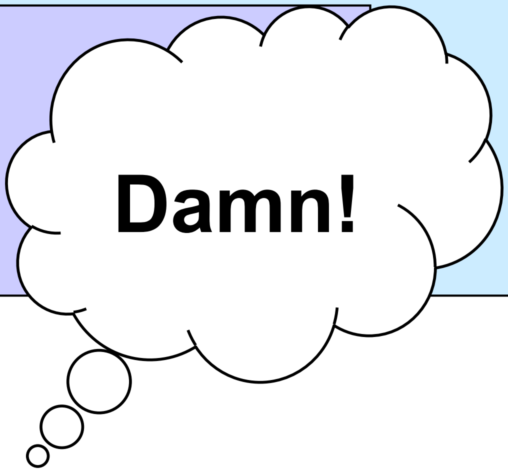
- Pointers are typed
  - the types of the objects they point to are known
  - there is one exception (discussed later)
- Pointers are first-class citizens
  - they can be passed to functions
  - they can be stored in arrays and other data structures
  - they can be returned by functions

Swapping

What does this do?

```c
void swap(int *i, int *j) {
    int *tmp;
    tmp = j; j = i; i = tmp;
}
int main() {
    int a = 4;
    int b = 8;
    swap(&a, &b);
    printf("a:%d  b:%d\n", a, b);
}
```

```
$ ./a.out
a:4  b:8
```

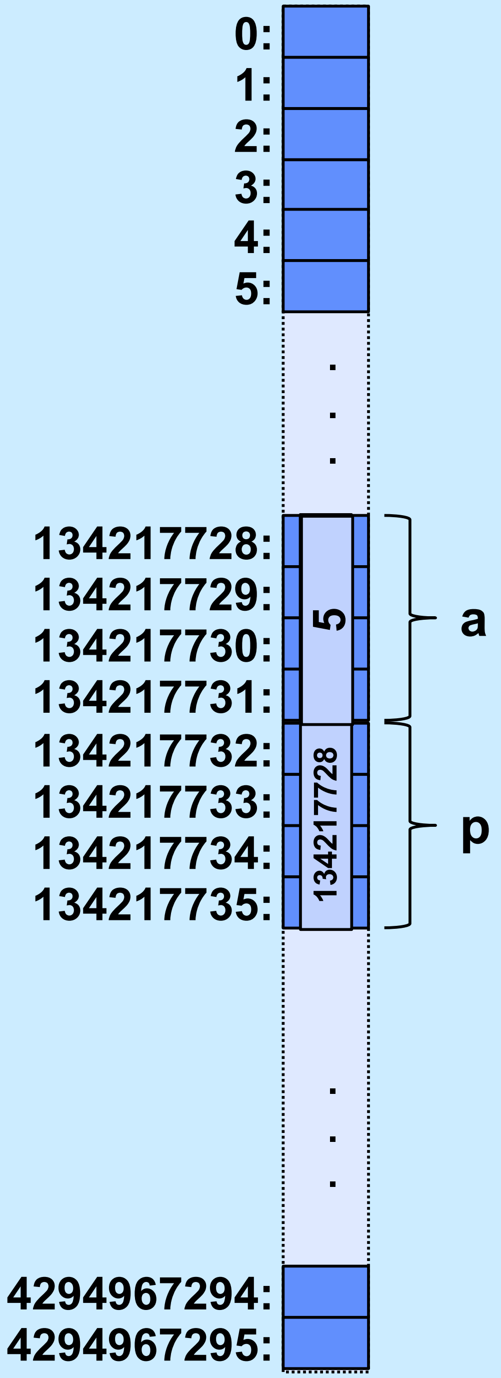


C Pointers

- **Dereferencing pointers**
  - **accessing/modifying the value pointed to by a pointer**

```
int main() {
    int *p;
    int a = 4;
    p = &a;
    printf("%d\n", *p);
    *p = *p + 1;
    printf("%d\n", *p);
}
```

```
$ ./a.out
4
5
```



Dereferencing C Pointers

```c
int main() {
   int *p;
   int a = 4;
   p = &a;
   printf("%d\n", *p);
   *p = *p + 1;
   *p += 3;
   printf("%d\n", a);
}
```

```
$ ./a.out
4
8
```

Swapping

```c
void swap(int *i, int *j) {
    int tmp;
    tmp = *j; *j = *i; *i = tmp;
}
int main() {
    int a = 4;
    int b = 8;
    swap(&a, &b);
    printf("a:%d  b:%d\n", a, b);
}
```

```
$ ./a.out
a:8  b:4
```

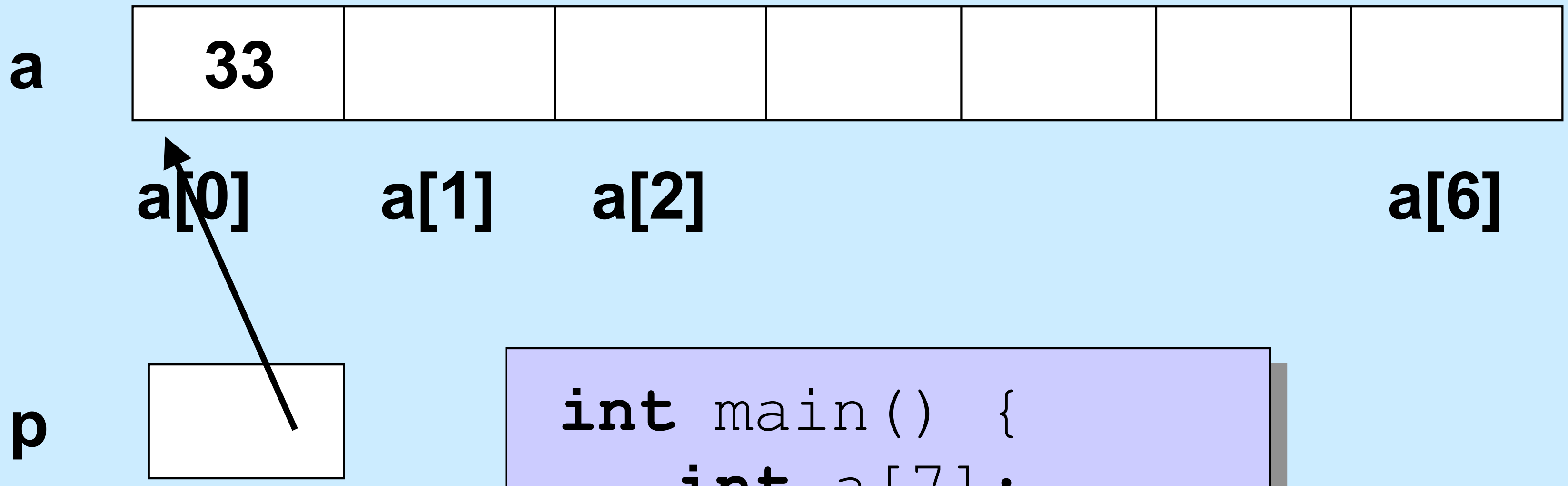
Quiz 1

```
int doubleit(int *p) {
   *p = 2*(*p);
   return *p;
}
int main() {
   int a = 3;
   int b;
   b = doubleit(&a);
   printf("%d\n", a*b);
}
```

What's printed?

a) 0
b) 12
c) 18
d) 36

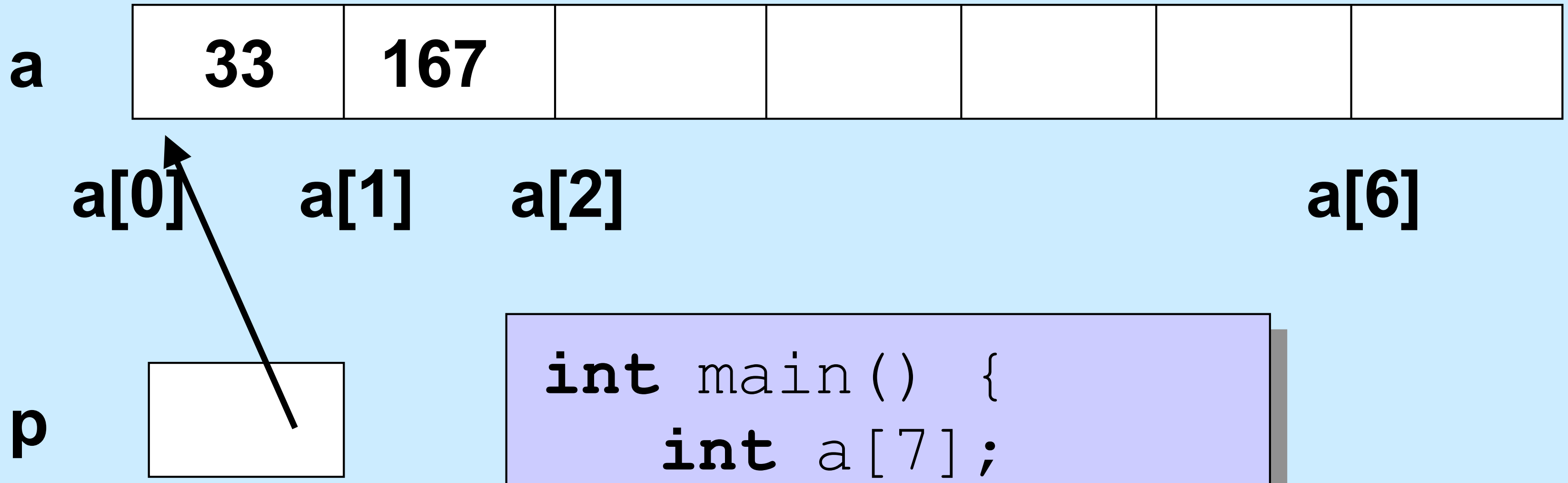
Pointers and Arrays



```
int main() {
    int a[7];
    int *p;
    p = &a[0];
    *p = 33;
}
```

Pointer Arithmetic

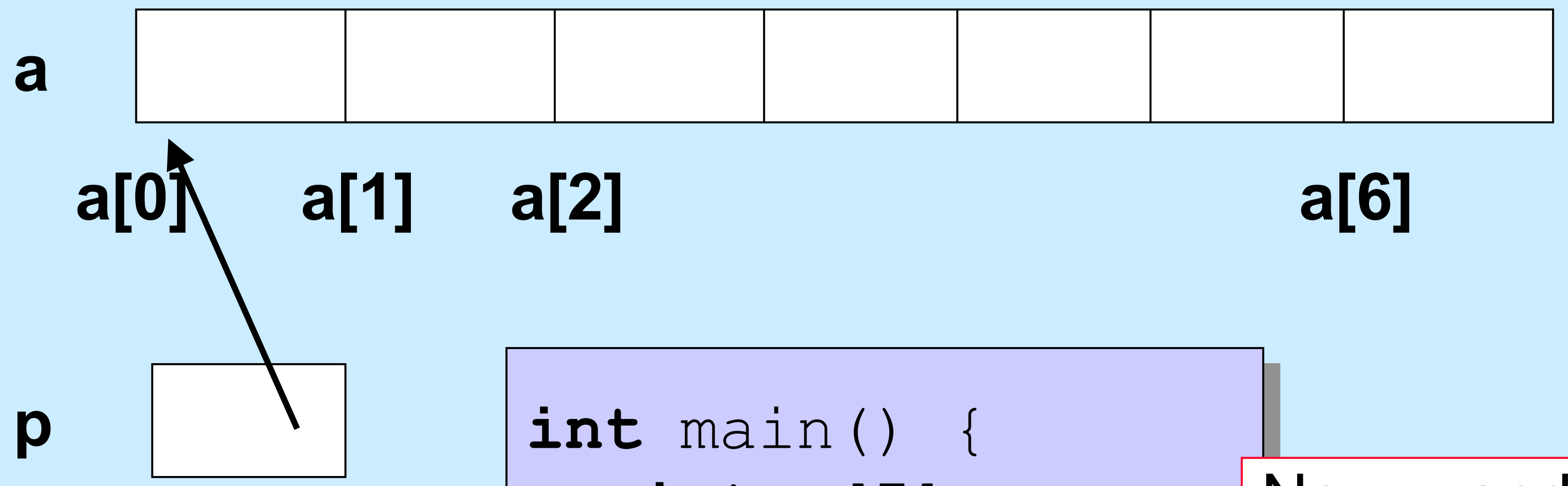
- **Pointers can be incremented/decremented**
  - what this does depends on its type



```
int main() {
    int a[7];
    int *p;
    p = &a[0];
    *p = 33;
    *(p+1) = 167;
}
```

Pointer Arithmetic

- **Pointers can be incremented/decremented**
  - what this does depends on its type

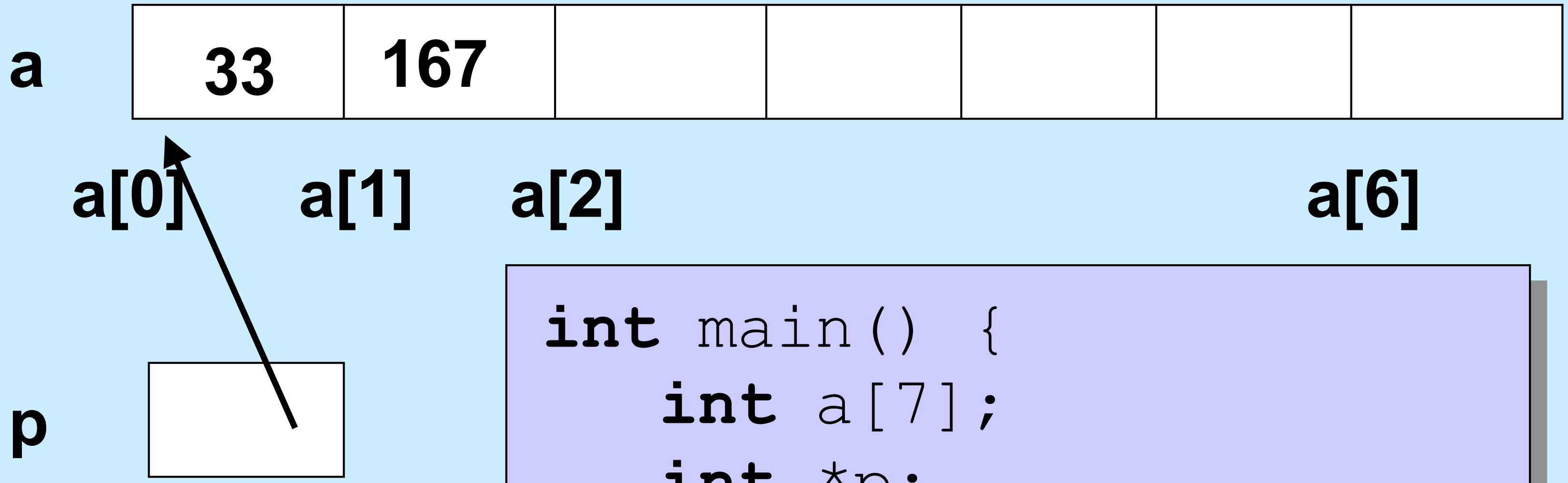


```
int main() {
    int a[7];
    int *p;
    p = &a[0];
}
```

Now `p` and `a` have the same value

Pointer Arithmetic

- **Pointers can be incremented/decremented**
  - **what this does depends on its type**



```
int main() {
    int a[7];
    int *p;
    p = a;
    *p = 33;
    p[1] = 167;
}
```

Pointers and Arrays

`p = &a[0];` can also be written as `p = a;`

`a[i];` really is `*(a+i)`

- This makes sense, yet is weird ...
  - `p` is of type `int *`
    - it can be assigned to

      ```
      int *q;
      p = q;
      ```
  - `a` sort of behaves like an `int *`
    - but it can't be assigned to

      ~~`a = q;`~~

Pointers and Arrays

- An array name represents a pointer to the first element of the array
- Just like a literal represents its associated value
  - in:

    ```
    x = y + 2;
    ```

    - “2” is a *literal* that represents the value 2
  - can’t do

    ```
    2 = x + y;
    ```

Literals and Functions

```
int func(int x) {
    x = x + 4;
    return x * 2;
}

int main() {
    result = func(2);
    printf("%d\n", result);
    return 0;
}
```

initialized with a copy of the argument

Arrays and Functions

initialized with a copy of the argument

```
int func(int *a, int nelements) {
    // sizeof(a) == sizeof(int *)
    int i;
    for (i=0; i<nelements-1; i++)
        a[i+1] += a[i];
    return a[nelements-1];
}

int main() {
    int array[50] = ... ;
    // sizeof(array) == 50*sizeof(int)
    printf("result = %d\n", func(array, 50));
    return 0;
}
```

Equivalently ...

```
int func(int a[], int nelements) {
    // sizeof(a) == sizeof(int *)
    ...

}

int main() {
    int array[50] = ... ;
    // sizeof(array) == 50*sizeof(int)
    printf("result = %d\n", func(array, 50));
    return 0;
}
```

No need for array size, since all that's used is pointer to first element

Quiz 2

```
int func(int a[], int nelements) {
   int b[5] = {0, 1, 2, 3, 4};
   a = b;
   return a[1];
}

int main() {
   int array[50];
   array[1] = 0;
   printf("result = %d\n",
      func(array, 50));
   return 0;
}
```

This program prints:

- a) 0
- b) 1
- c) 2
- d) nothing: it doesn’t compile because of a syntax error

Quiz 3

```
int func(int a[], int nelements) {
   int b[5] = {0, 1, 2, 3, 4};
   a = b;
   return a[1];
}

int main() {
   int array[5] = {4, 3, 2, 1, 0};
   func(array, 5);
   printf("%d\n", array[1]);
   return 0;
}
```

This program prints:

a) 0
b) 1
c) 2
d) 3