APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 16
February 27, 2012

Today

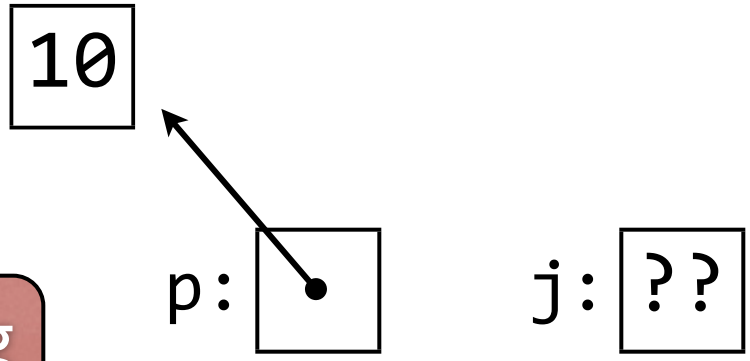
- Even More Pointers
- Pointers and Arrays

The Most Common Mistake

- Using an uninitialized pointer

```
int *p;
int j = *p;
```

Wrong




```
*p = 10;
```

Very Wrong

Pointer Fun with Binky

by Nick Parlante

This is document 104 in the Stanford CS Education Library — please see cslibrary.stanford.edu for this video, its associated documents, and other free educational materials.

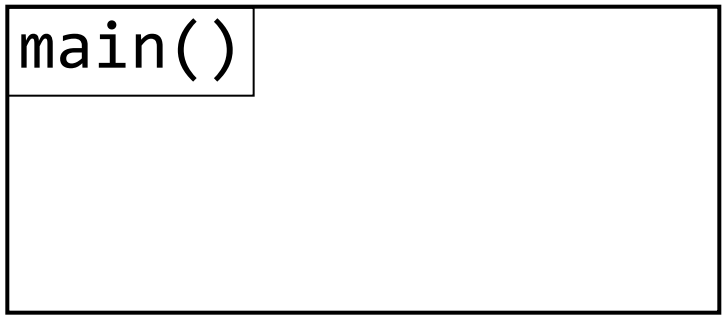
Carpe Post Meridiem!

Returning Pointers

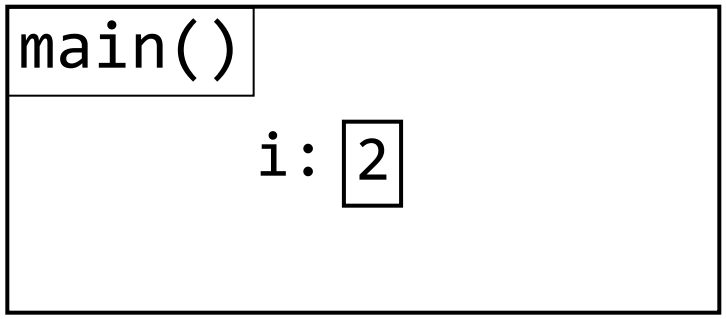
```
int *max(int *p, int *q)
{
    if (*p > *q)
    {
        return p;
    }
    else
    {
        return q;
    }
}
```

```
int main(void)
{

    int i = 8;
    int j = 9;

    int *r;

    r = max(&i, &j);

    printf("%d\n", *r);

    return 0;
}
```

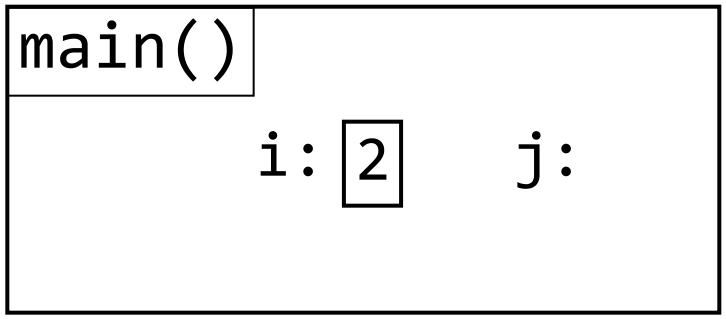
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



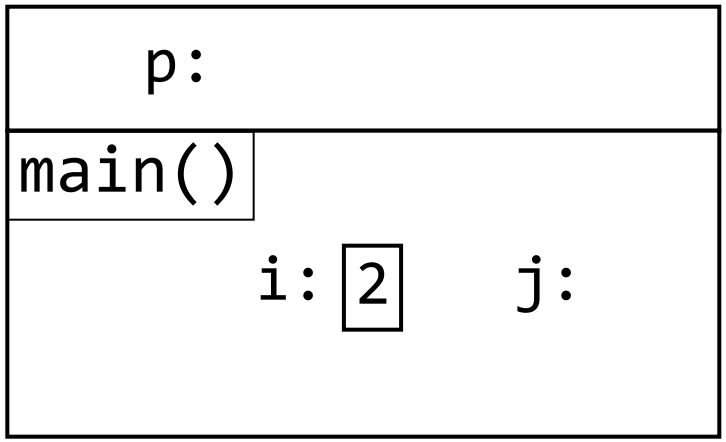
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



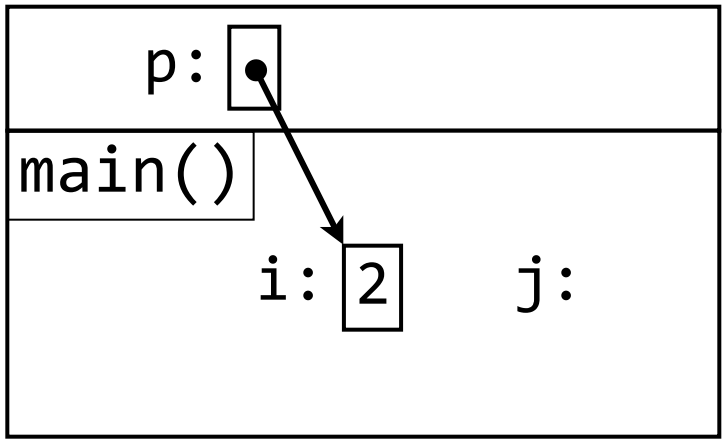
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



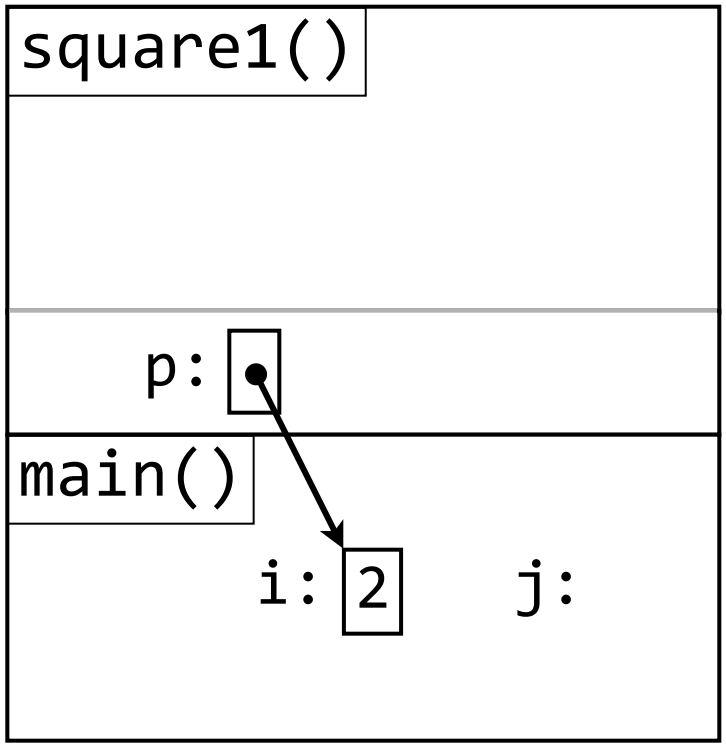
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



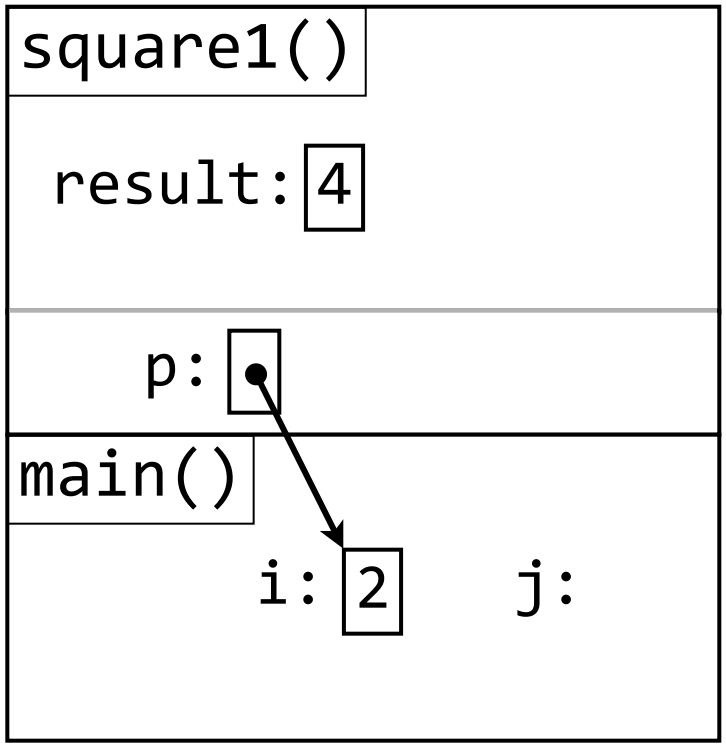
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



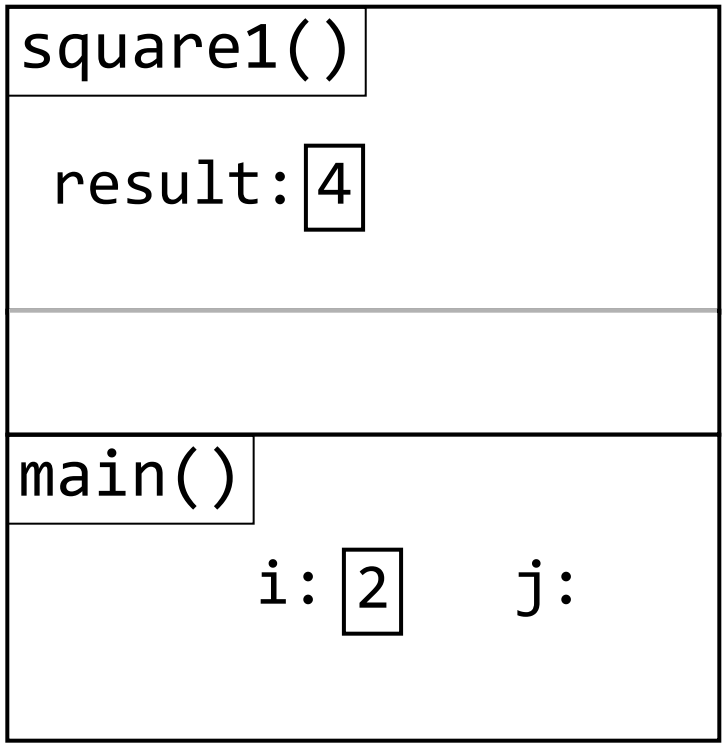
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



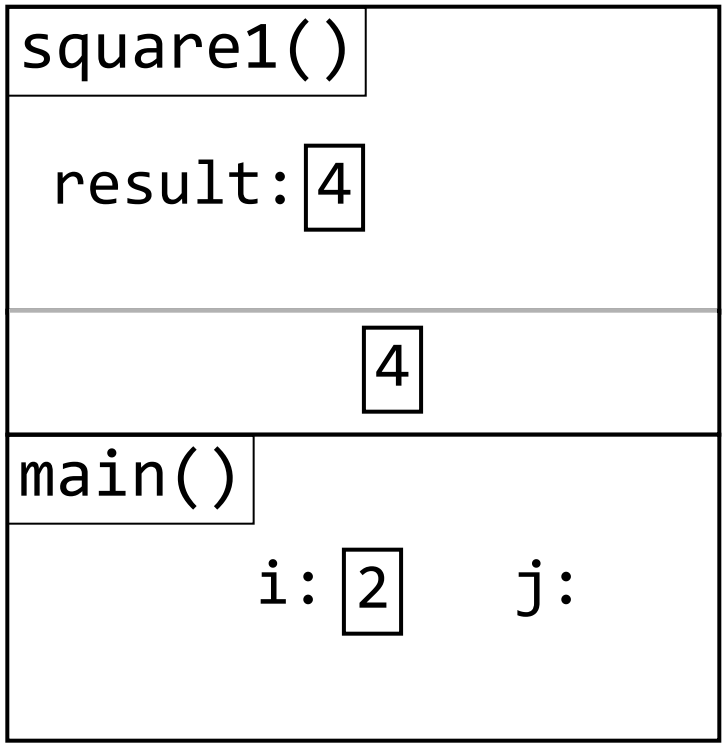
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



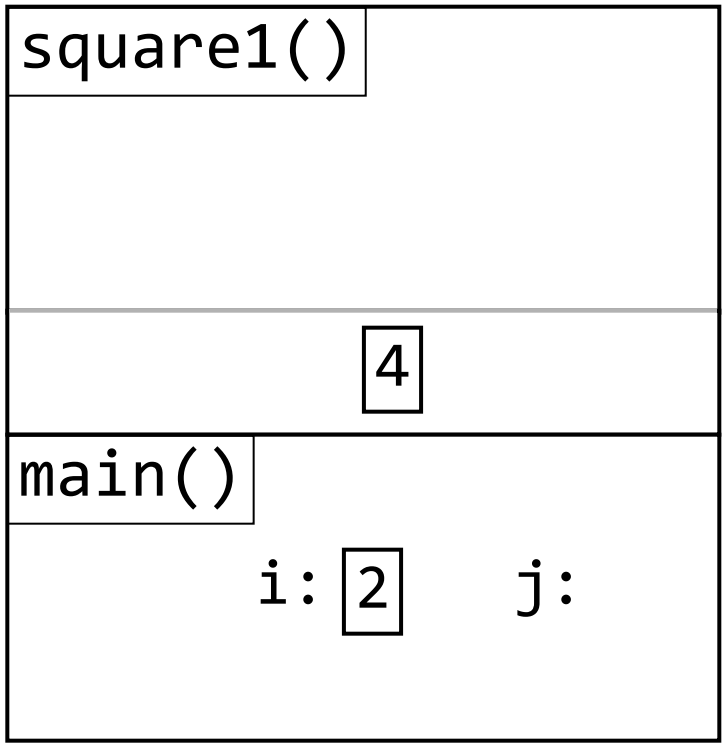
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



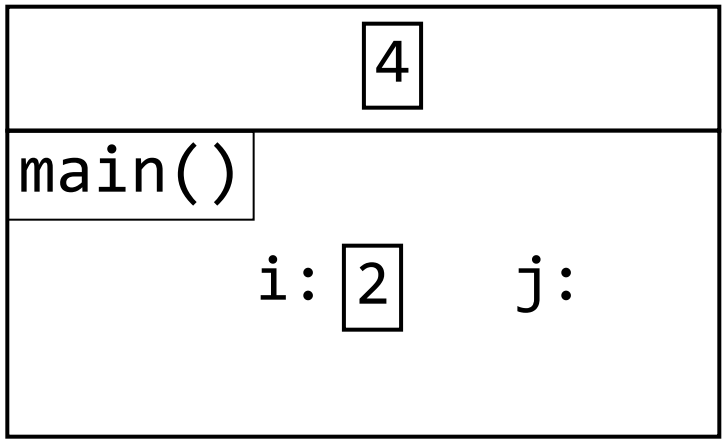
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



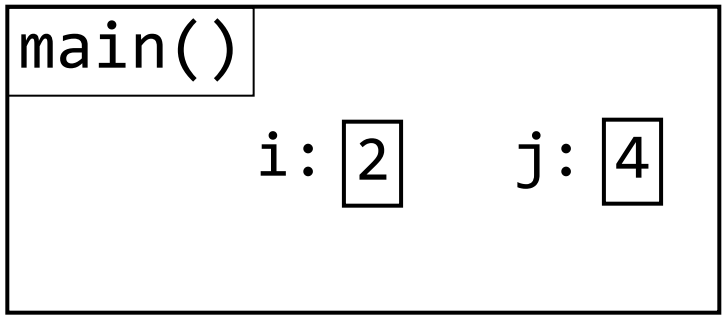
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



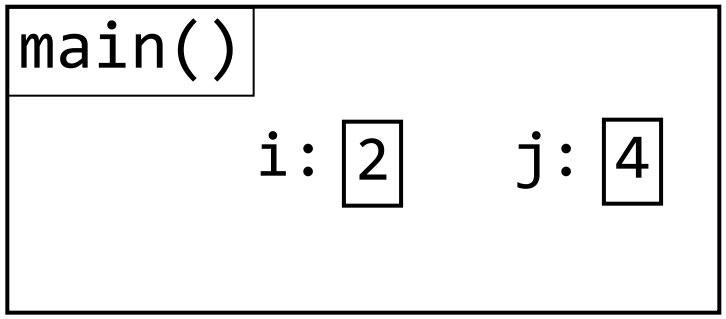
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



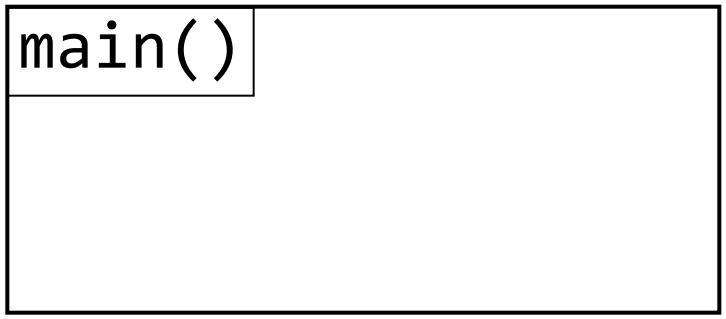
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);

  return 0;
}
```



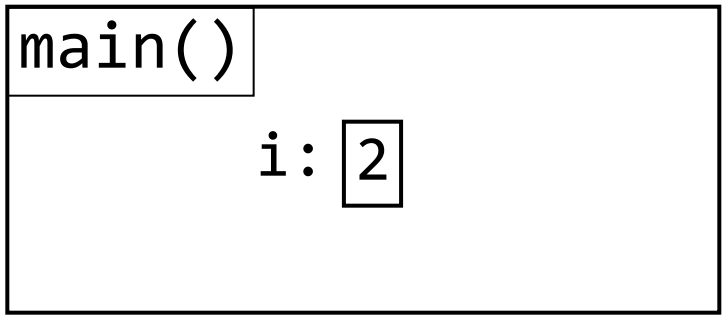
```
int square1(int *p)
{
  int result = *p * *p;
  return result;
}

int main(void)
{
  int i = 2;
  int j = square1(&i);
  printf("%d", j);
                    4
  return 0;
}
```



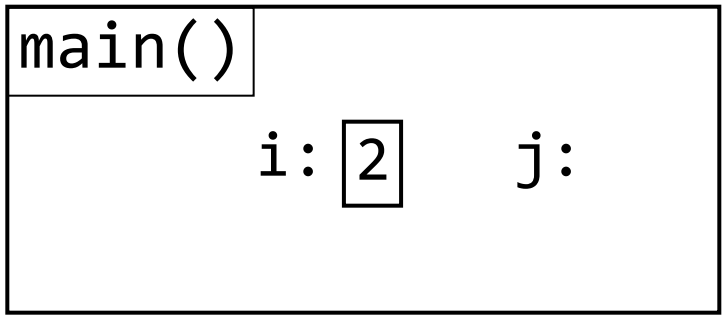
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



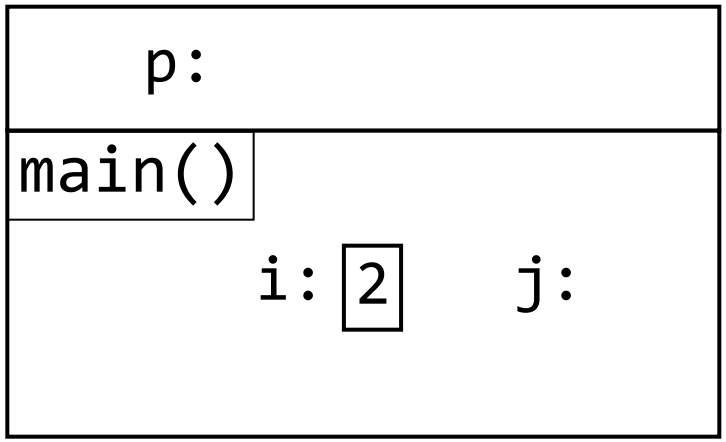
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



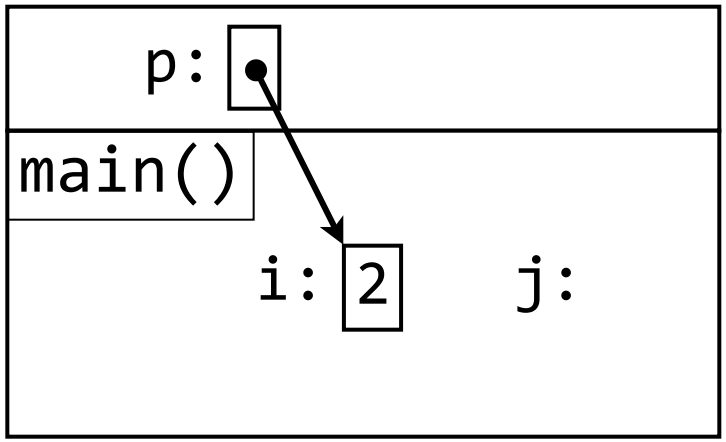
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



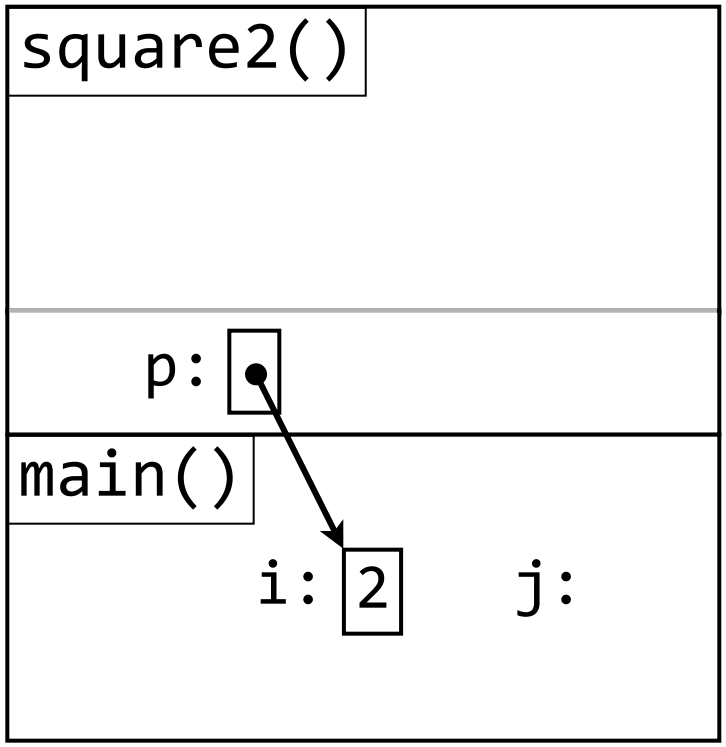
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



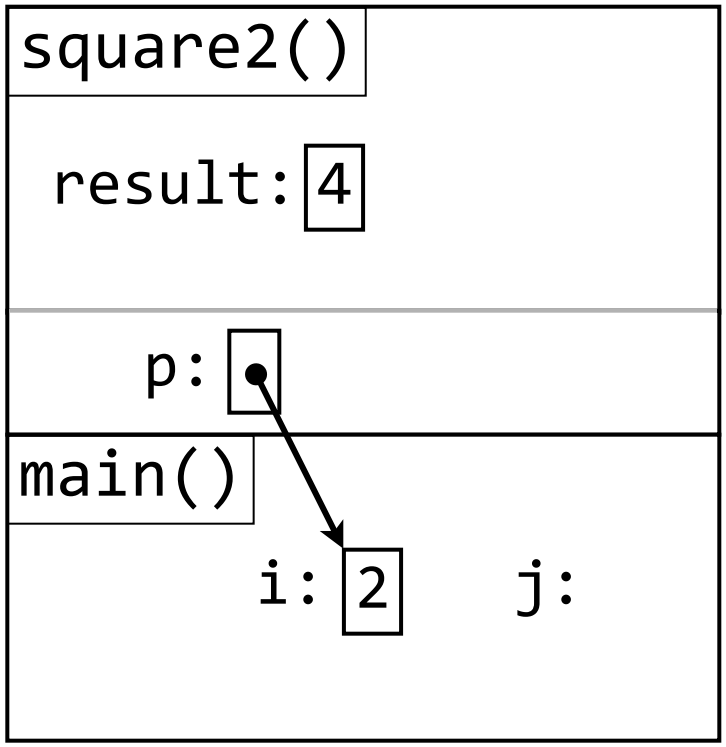
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



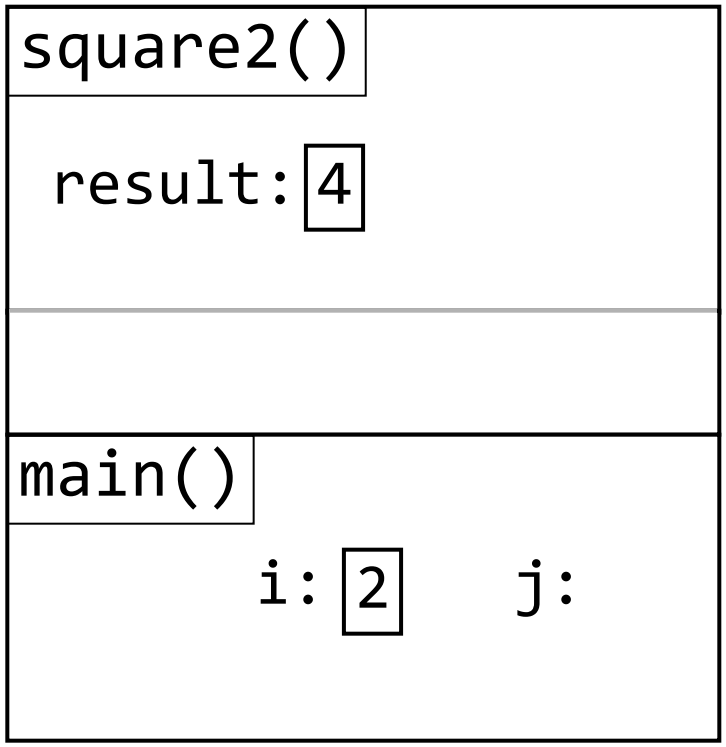
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



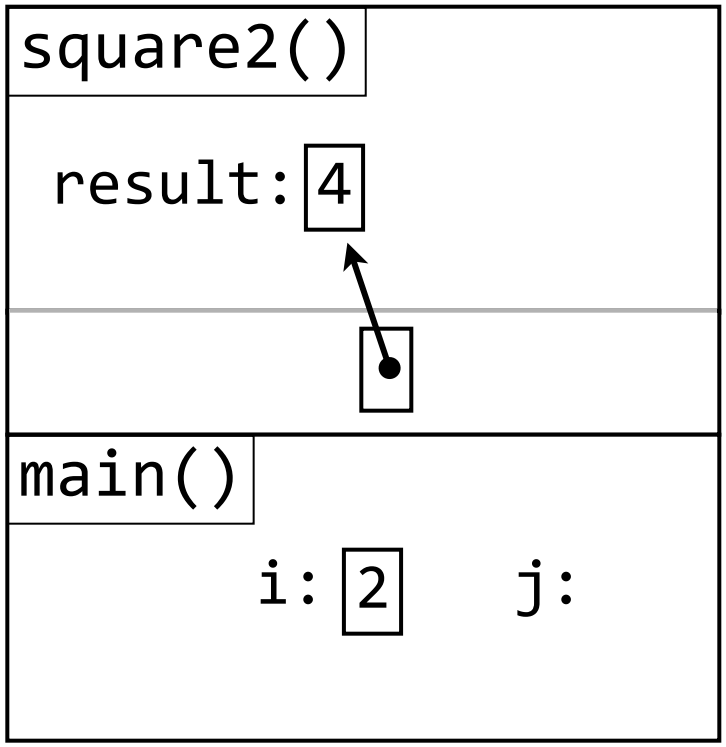
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



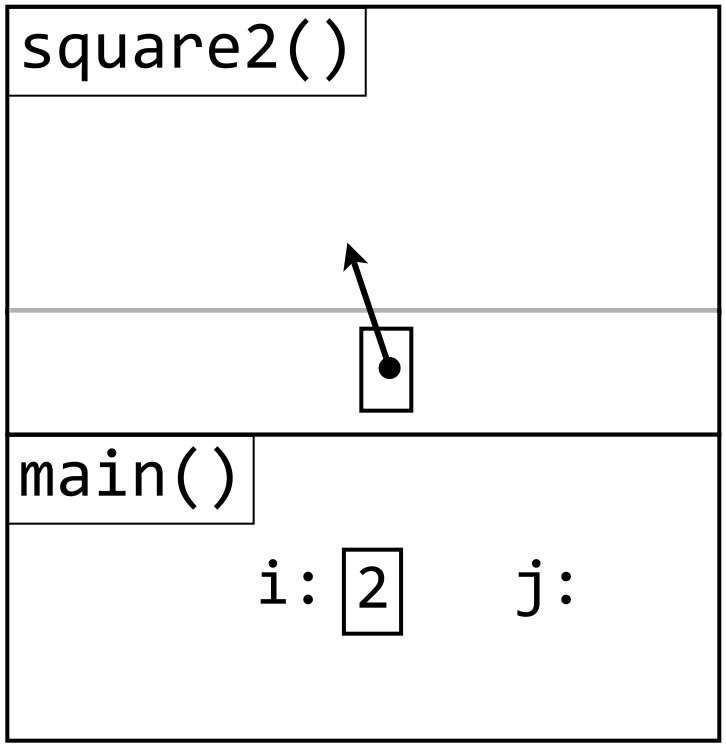
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



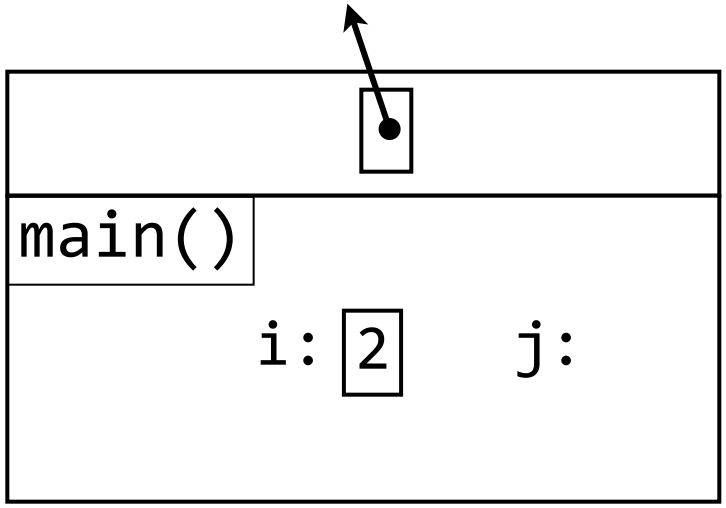
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



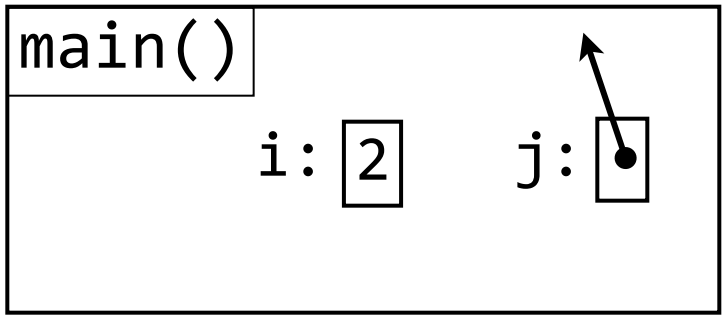
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



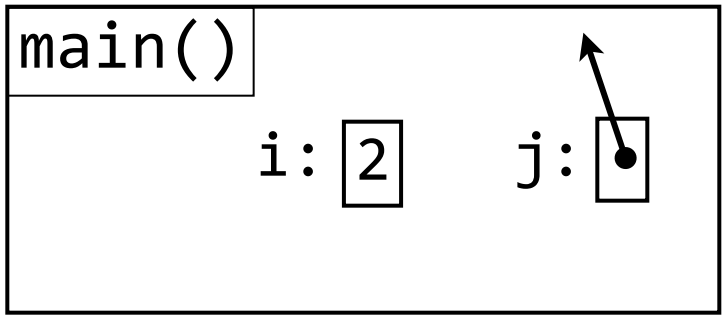
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```



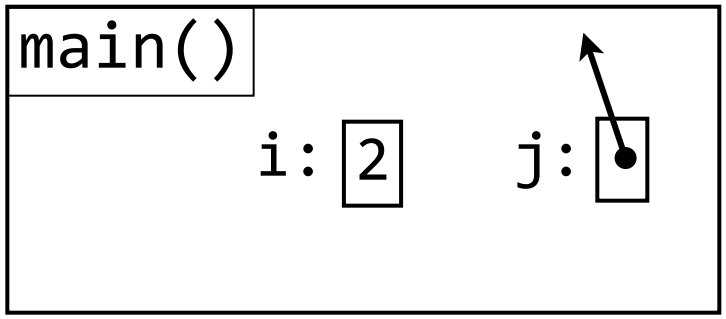
```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);
                  ????
  return 0;
}
```



```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main(void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```

Wrong

????



Pointing to Nothing

- You can't follow a pointer that points at something that doesn't exist anymore!
  - (Well, you can, but it's undefined behaviour)
  - Same as using an uninitialized pointer

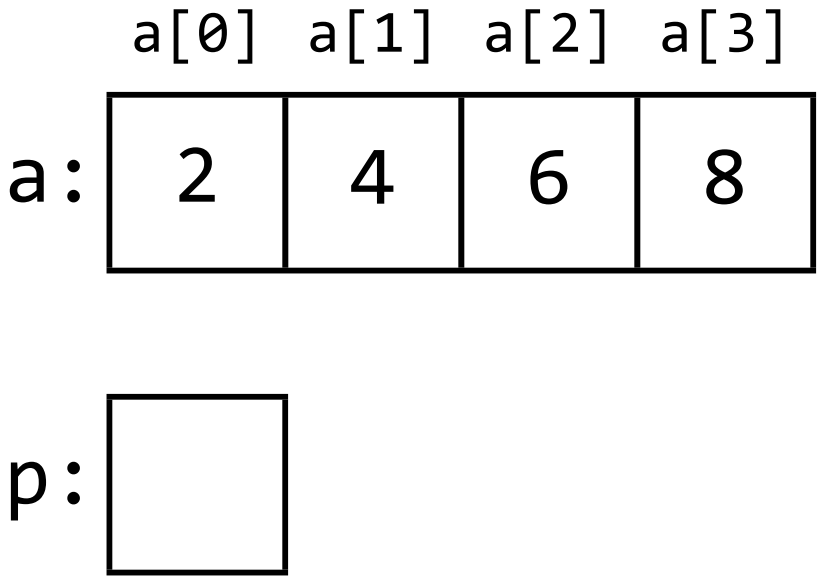
```
int *p;
*p = 42;
```

- Common source of this bug is via `return` from a function call

Pointers and Arrays

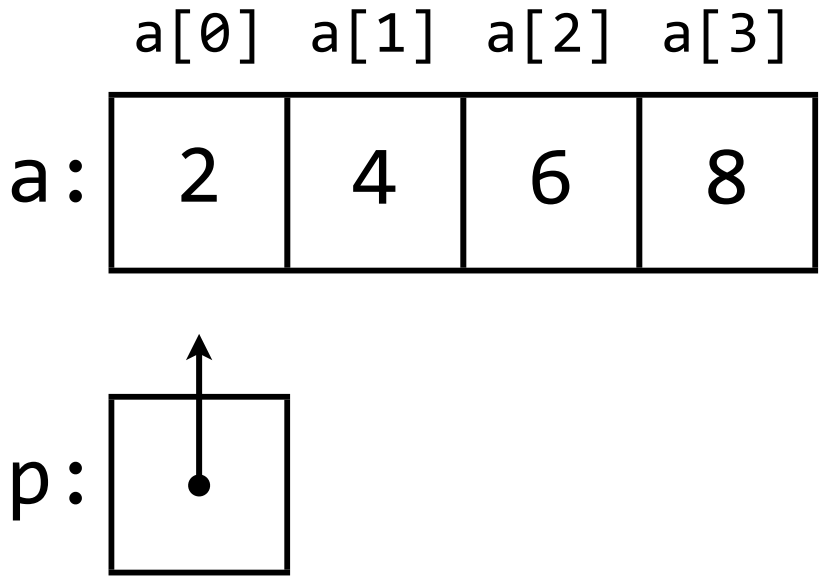
Pointing to Array Elements

```
int a[] = {2, 4, 6, 8};
int *p;
```



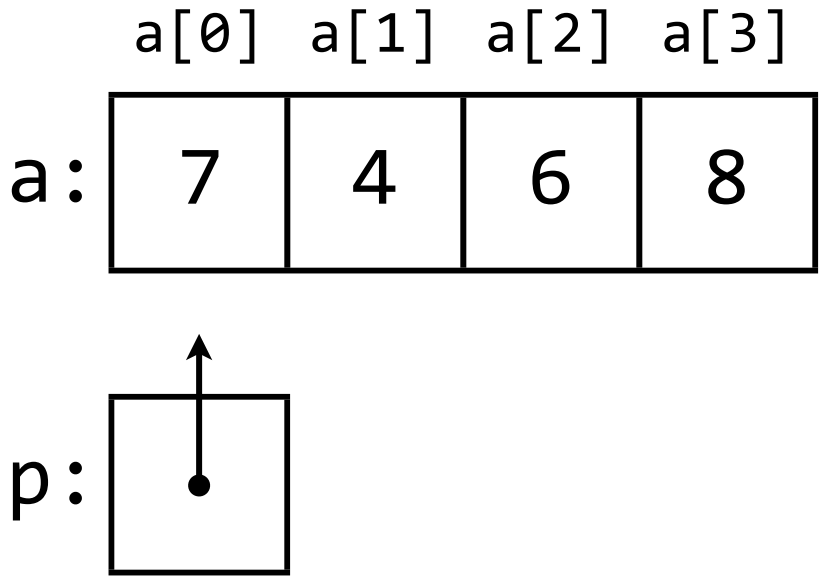
Pointing to Array Elements

```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
```



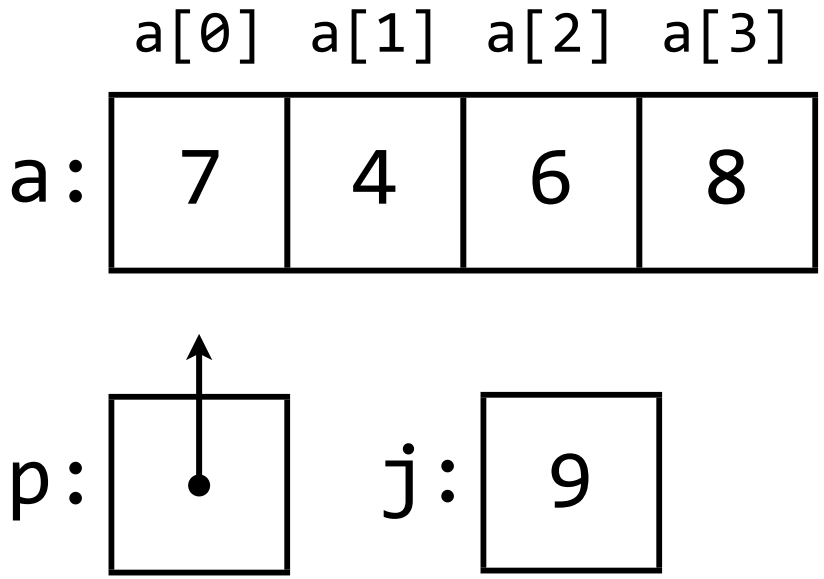
Pointing to Array Elements

```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
*p = 7;
```



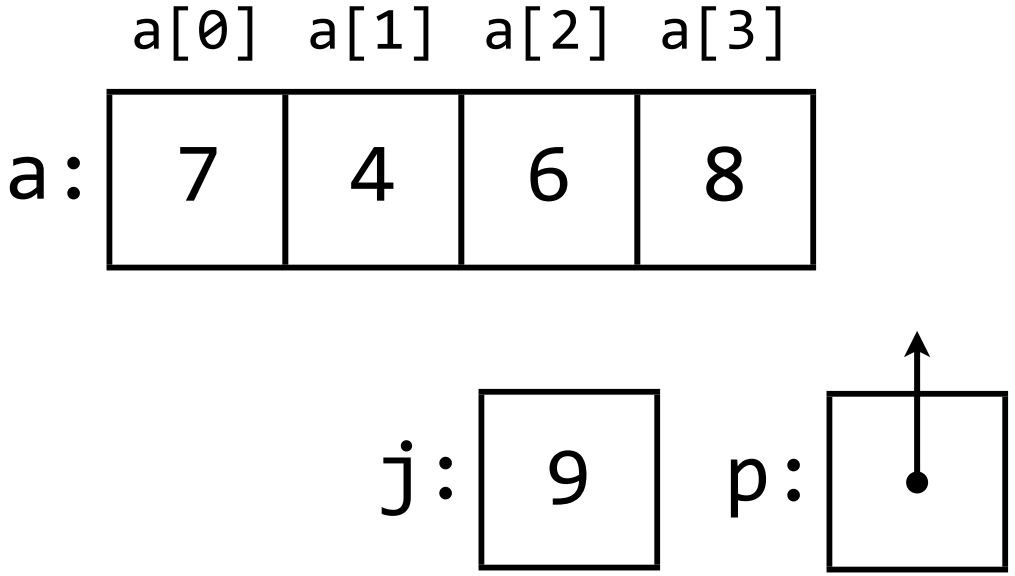
Pointing to Array Elements

```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
*p = 7;
int j = 2 + *p;
```



Pointing to Array Elements

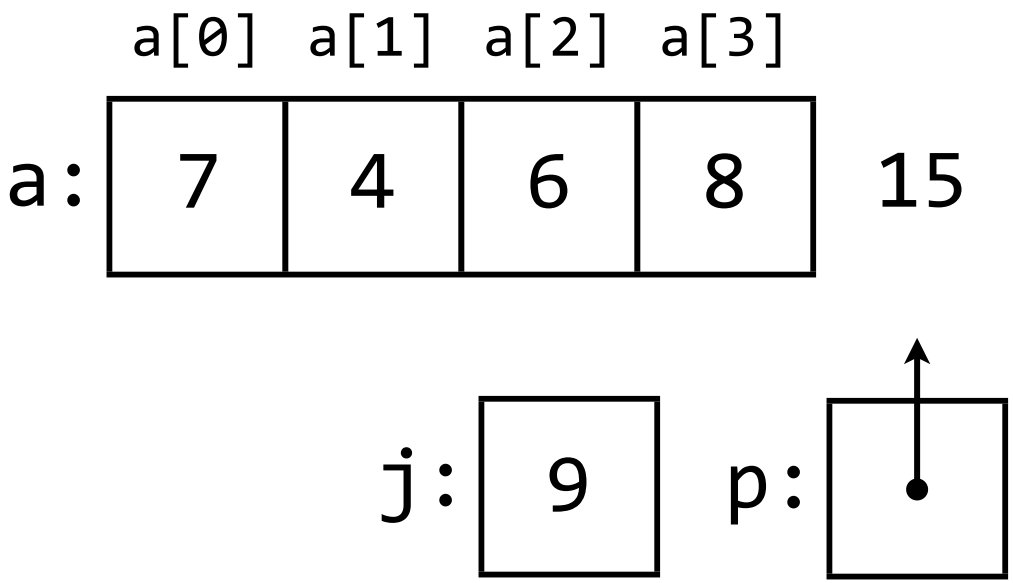
```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
*p = 7;
int j = 2 + *p;

p = &a[4];
```



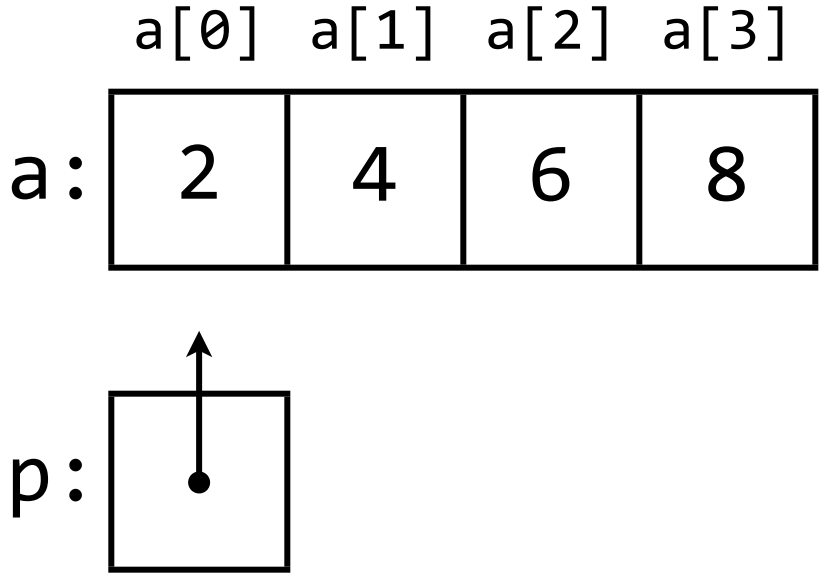
Pointing to Array Elements

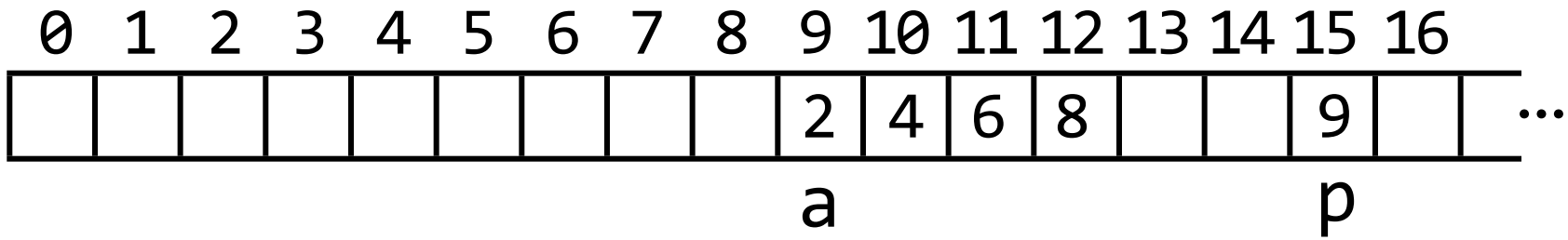
```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
*p = 7;
int j = 2 + *p;

p = &a[4];
*p = 15;
```

Wrong

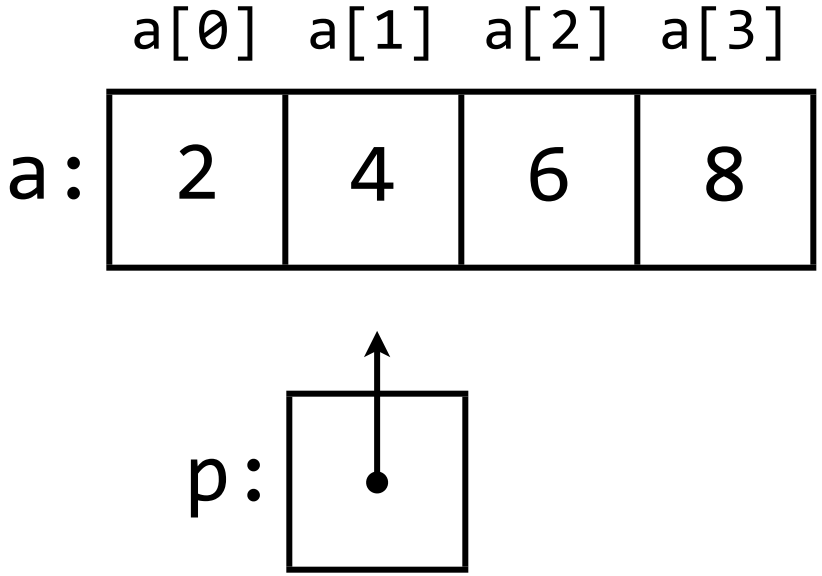


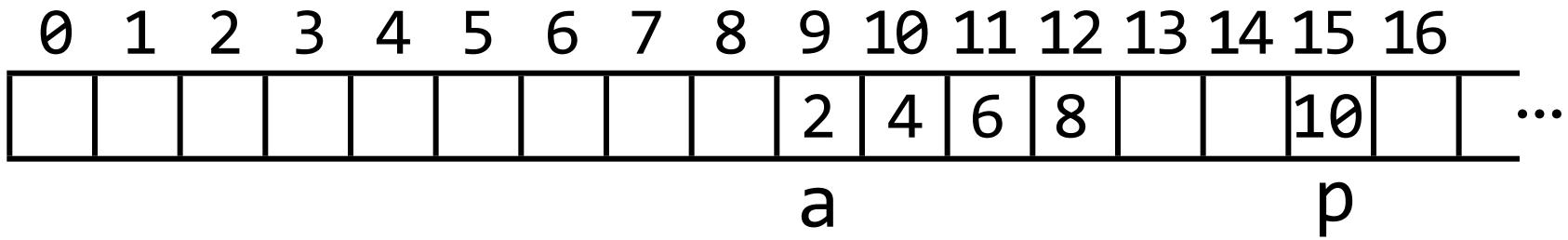
```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
```





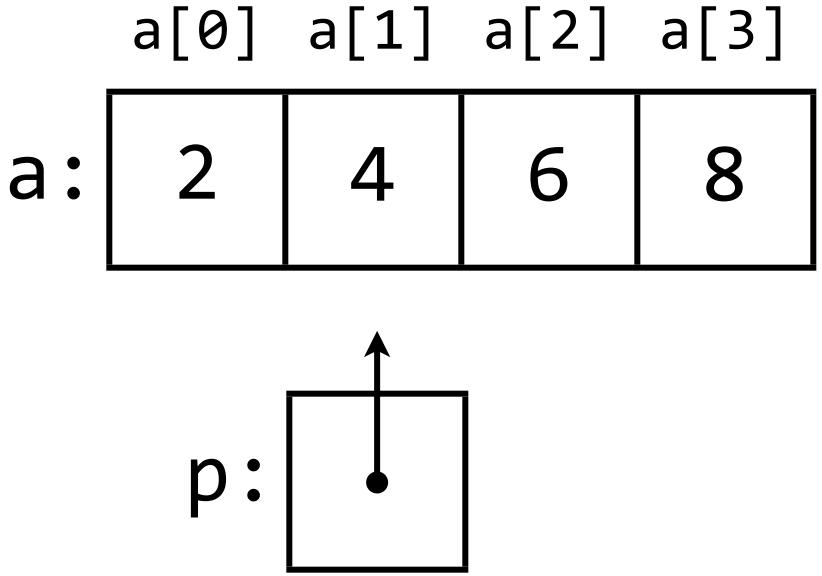
```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
p = &a[1];
```

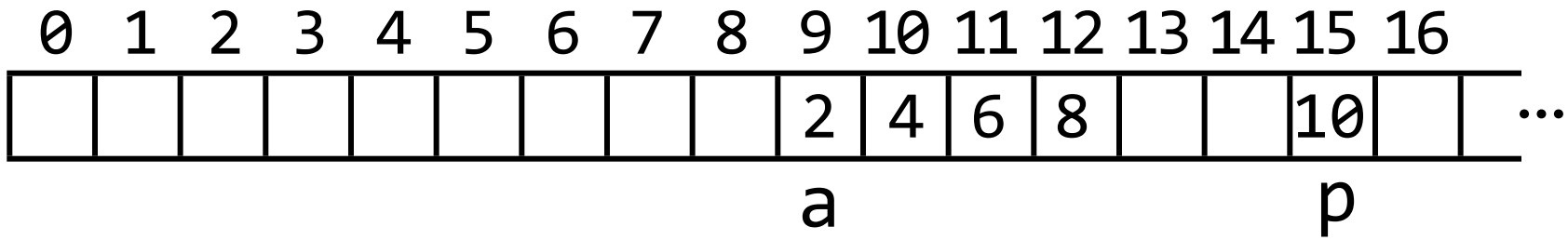




```
&a[1] is &a[0] + 1
&a[2] is &a[0] + 2
&a[i] is &a[0] + i
```

```
int a[] = {2, 4, 6, 8};
int *p;

p = &a[0];
p = &a[1];
```



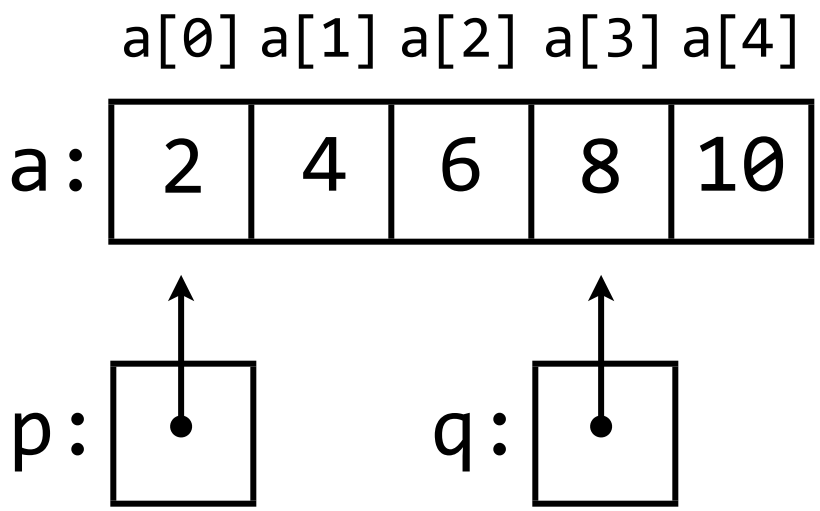


Pointer Arithmetic

- C let's us do (a few) arithmetic operations on pointers
  - Add an `int` and a pointer
  - Subtract an `int` from a pointer
  - Subtract one pointer from another

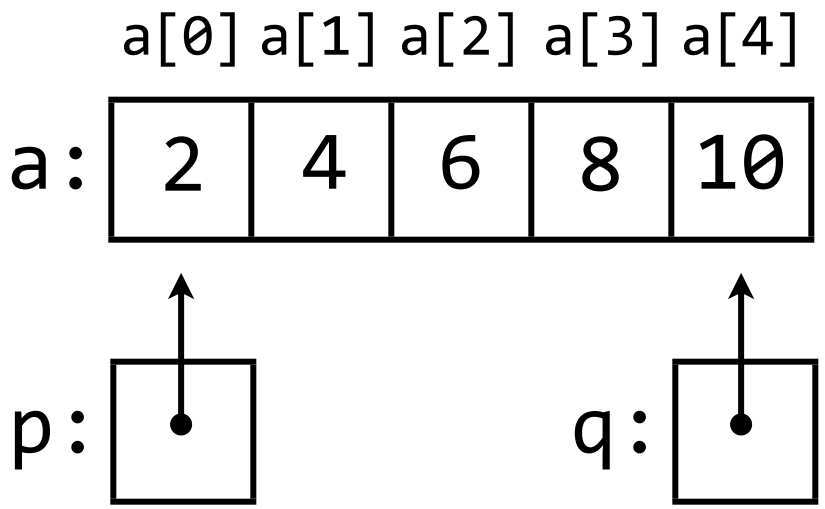
- These are only allowed within an array

```
int a[] = {2, 4, 6, 8, 10};
int *p;
int *q;

p = &a[0];
q = p + 3;
```



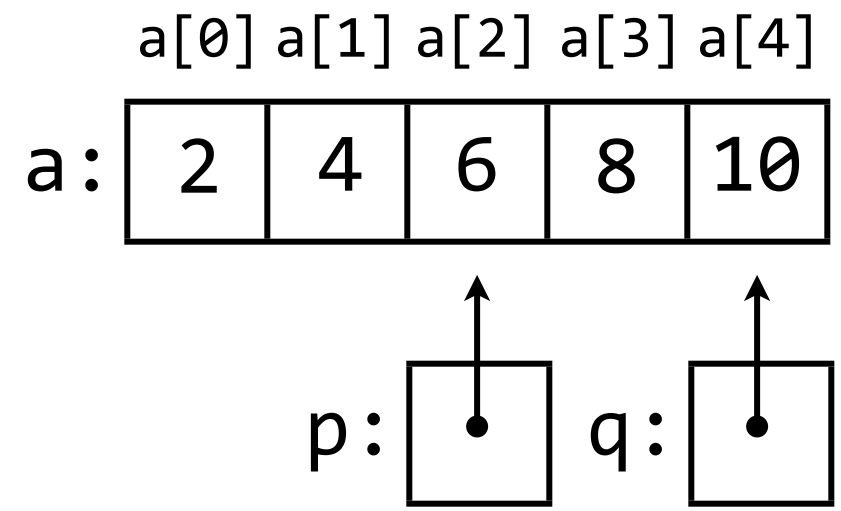
if p points to a[i]
then p + j points to a[i + j]

```
int a[] = {2, 4, 6, 8, 10};
int *p;
int *q;

p = &a[0];
q = p + 3;
q++;
```



if p points to a[i]
then p + j points to a[i + j]

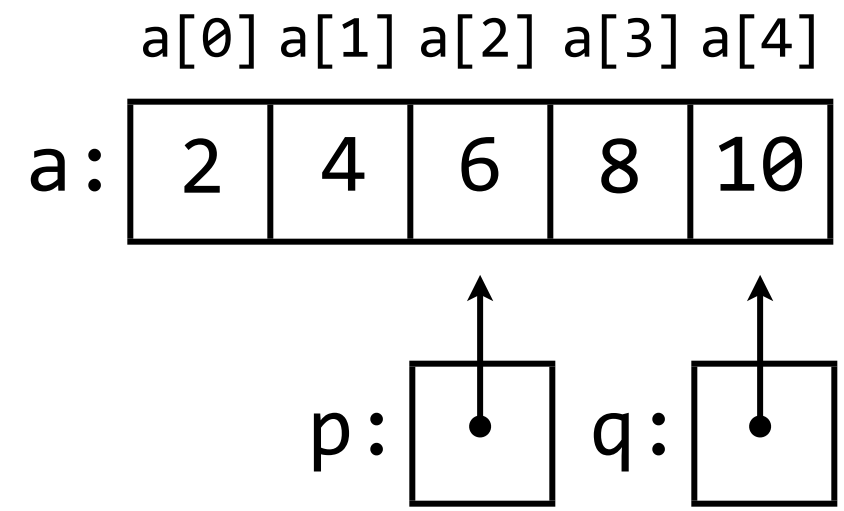
```
int a[] = {2, 4, 6, 8, 10};
int *p;
int *q;

p = &a[0];
q = p + 3;
q++;

p = q - 2;
```



if p points to a[i]
then p + j points to a[i + j]

if p points to a[i]
then p - j points to a[i - j]

```
int a[] = {2, 4, 6, 8, 10};
int *p;
int *q;

p = &a[0];
q = p + 3;
q++;

p = q - 2;

int distance = p - q;
```

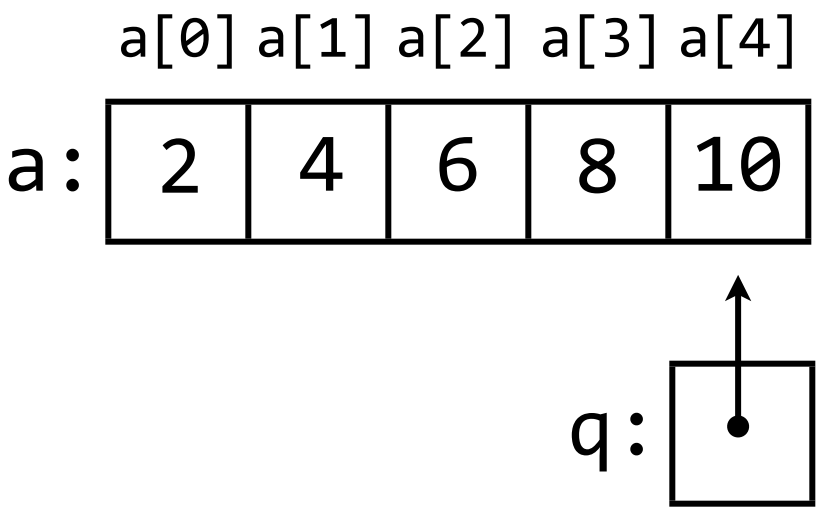


if p points to a[i]
then p + j points to a[i + j]

if p points to a[i]
then p - j points to a[i - j]

if p points to a[i] and q points to a[j]
then p - q is equal to i - j

-2




```
int a[] = {2, 4, 6, 8, 10};
int *p;
int *q;
```

```
p = &a[0];
q = p + 3;
q++;
```

if p points to a[i]
then p + j points to a[i + j]

```
p = q - 2;
```

if p points to a[i]
then p - j points to a[i - j]

```
int distance = p - q;
```

-2

if p points to a[i] and q points to a[j]
then p - q is equal to i - j

```
p -= 6;
```

Wrong

Pointer Arithmetic

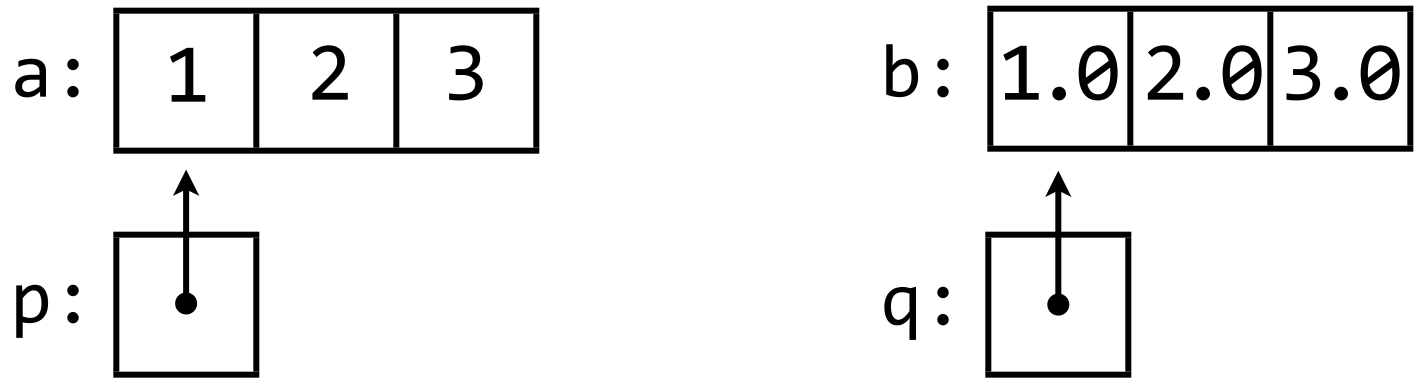
- Only allowed within an array

```
int i = 9;
int j = 12;

int *p = &i;
int *q = &j;

p += 2;
q  = p - 1;
int distance = q - p;
```

Wrong

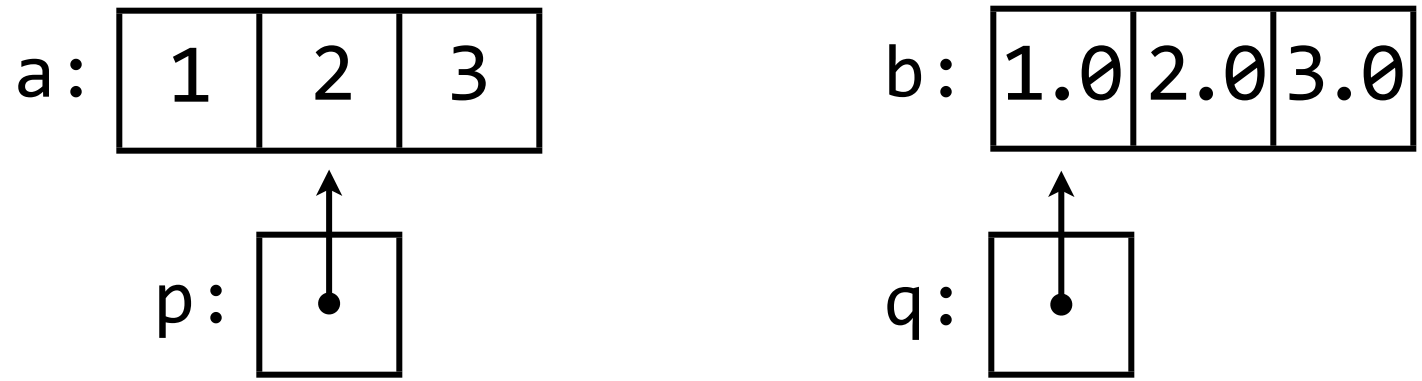
Scaling

```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```



Scaling

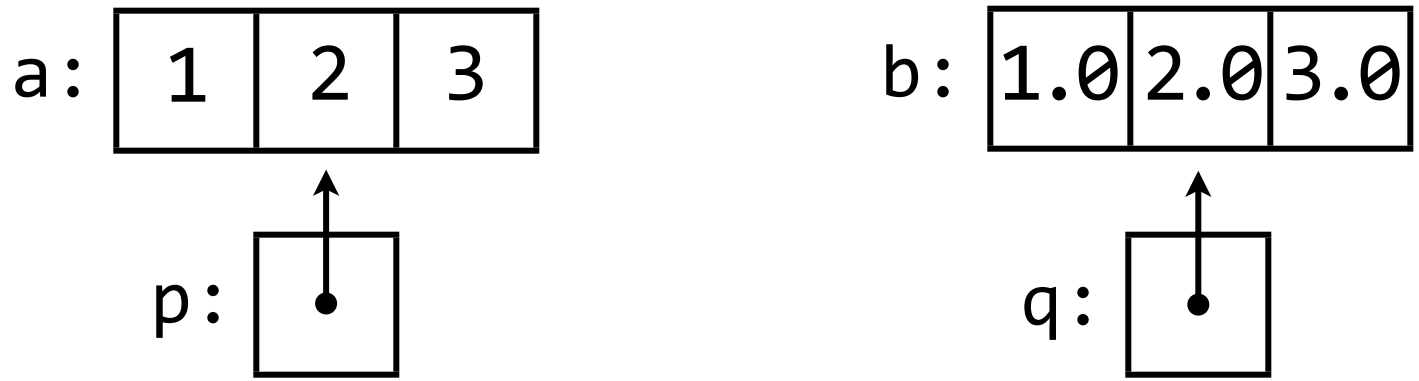
```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```




```
p++;
```

Scaling

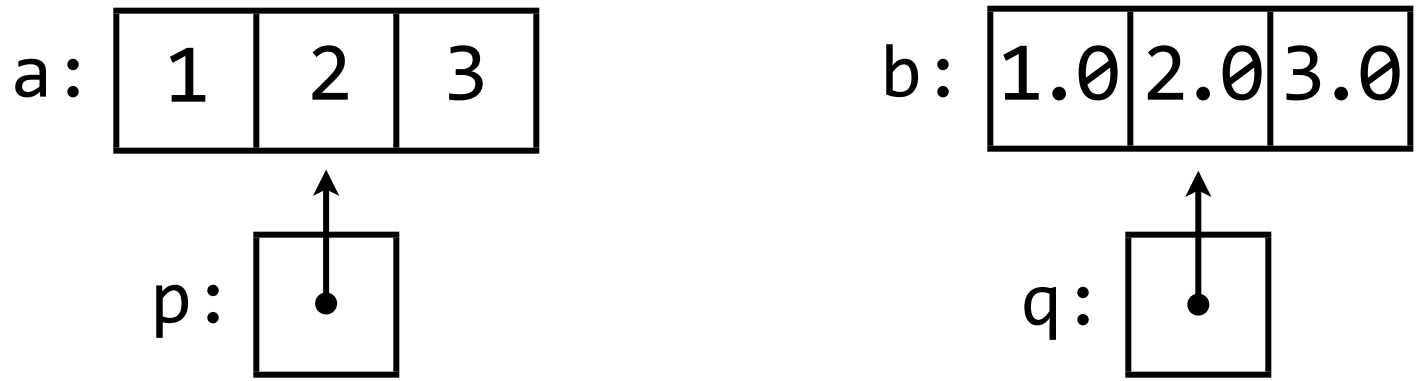
```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```




```
p++;
q++;
```

Scaling

```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```



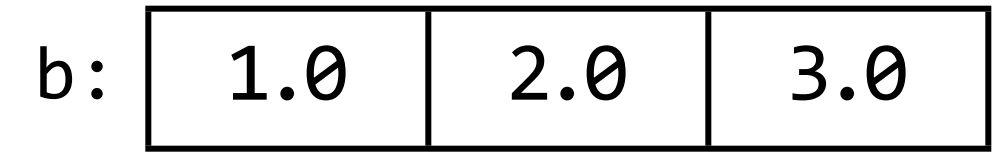

```
p++;
q++;
```

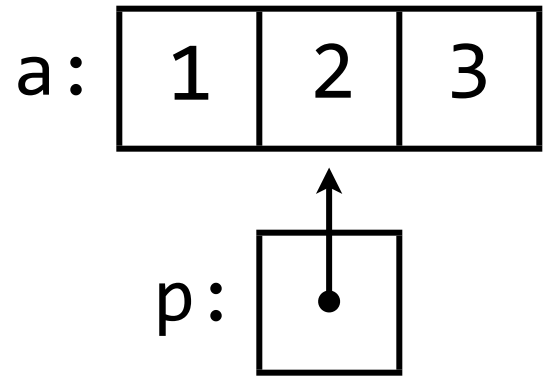
`int` is 4 bytes on ECF

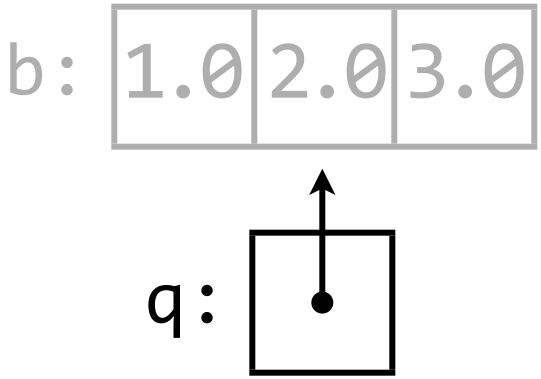
`double` is 8 bytes on ECF

Scaling

```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```







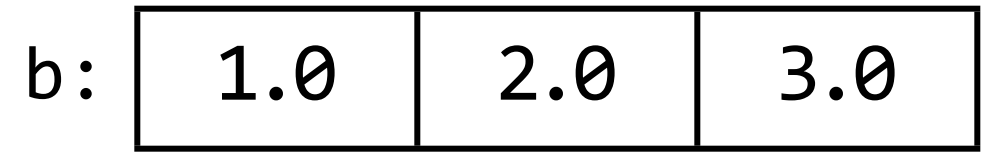

```
p++;
q++;
```

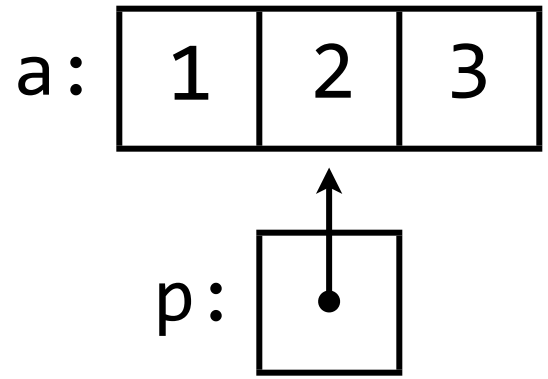
`int` is 4 bytes on ECF

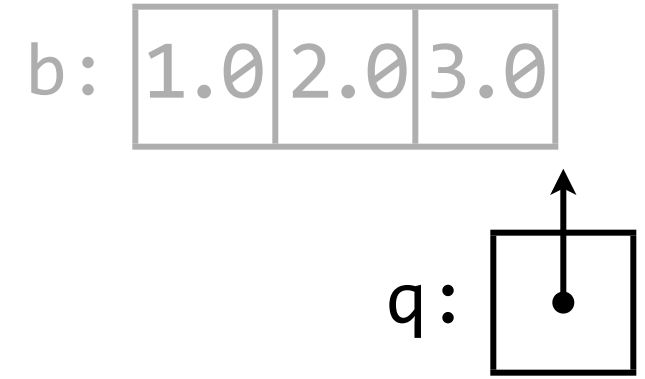
`double` is 8 bytes on ECF

Scaling

```
int a[] = {1, 2, 3};
double b[] = {1.0, 2.0, 3.0};
int *p = &a[0];
double *q = &b[0];
```








```
p++;
q++;
```

`int` is 4 bytes on ECF

`double` is 8 bytes on ECF

Comparing Pointers

- You can use relational operators (<, <=, >=, and >) on array element pointers
- They compare the indices, not the values

```
int a[] = {10, 9, 8, 7};
int *p = &a[0];
int *q = &a[3];
```

```
                0   3
bool b = p < q;
```

true

```
                10    7
bool b = *p < *q;
```

false

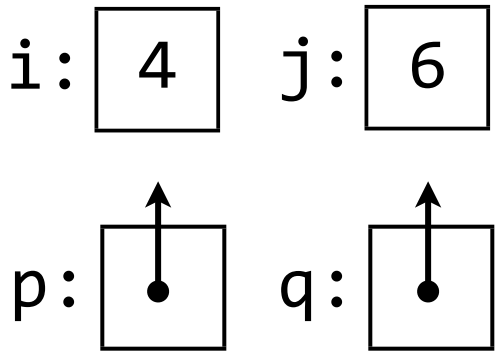
- You can compare non-array element pointers, but it's meaningless (i.e., legal but useless)

Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

false



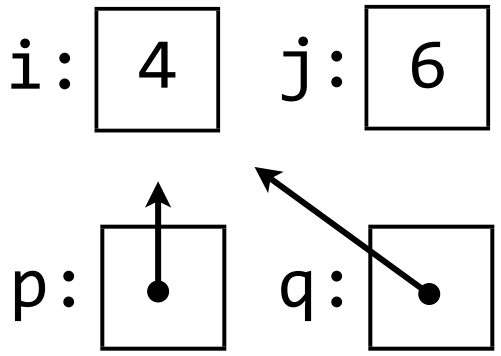
Equality

- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

false

```
q = &i;
```



Equality

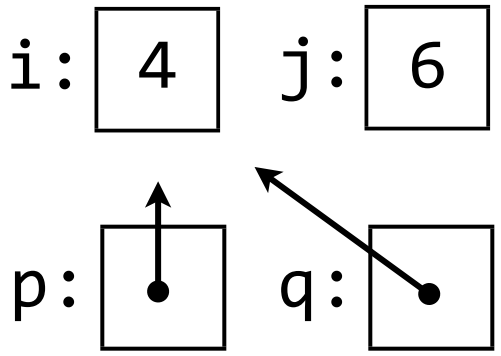
- You can use the comparison operators (== and !=) on any pointer
- They compare the arrows

```
int i = 4;
int j = 6;

int *p = &i;
int *q = &j;

bool b = (p == q);
```

false

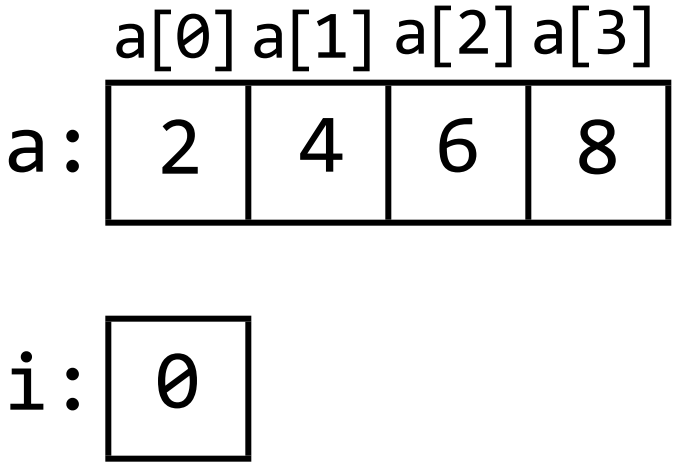
```
q = &i;
b = (p == q);
```

true



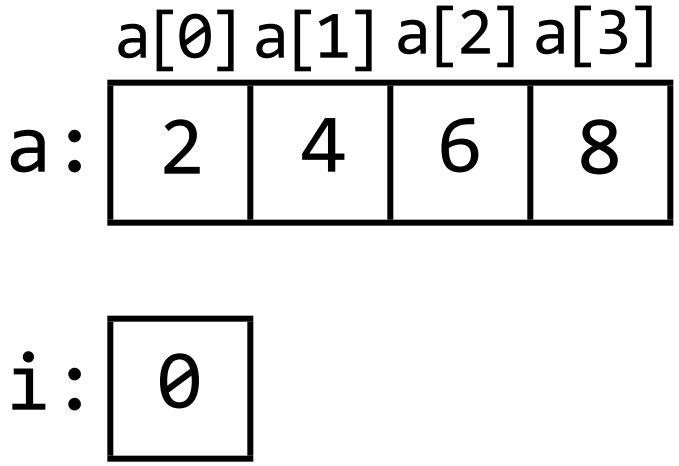
Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



Traversing Arrays

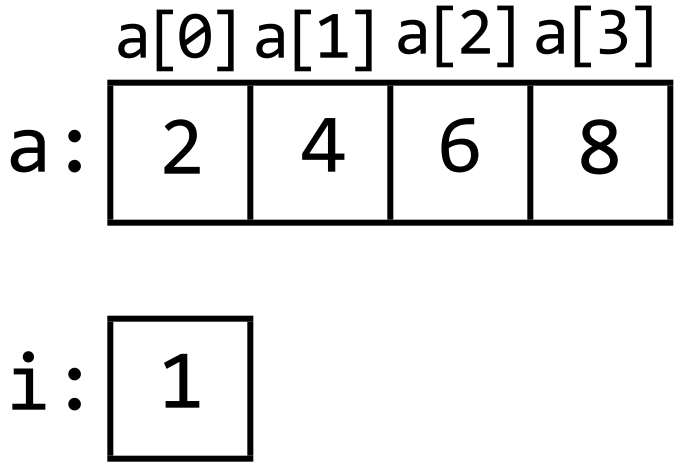
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



2

Traversing Arrays

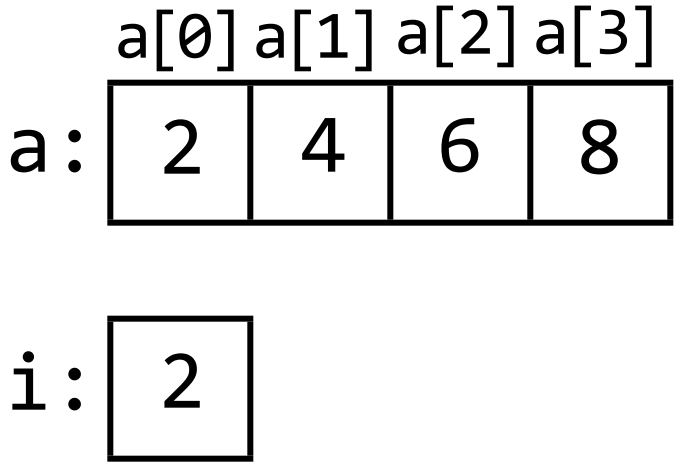
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



2 4

Traversing Arrays

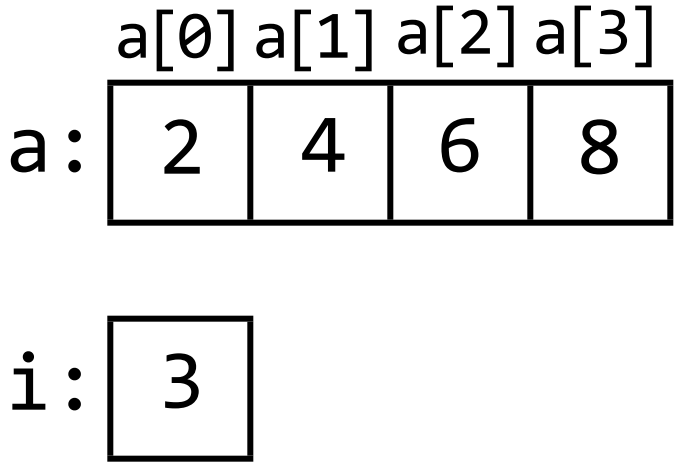
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



2 4 6

Traversing Arrays

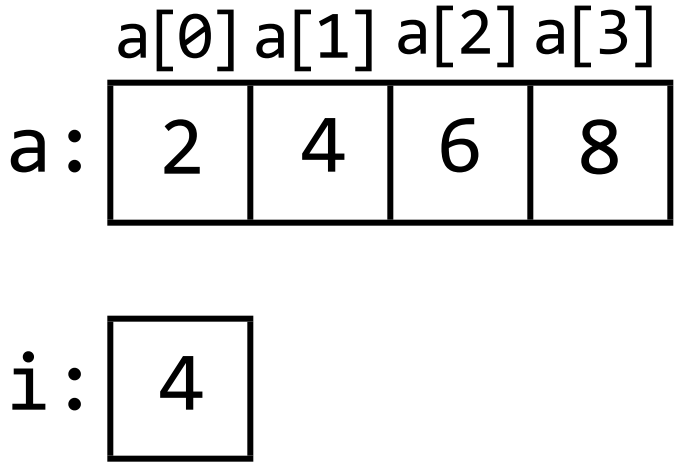
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



2 4 6 8

Traversing Arrays

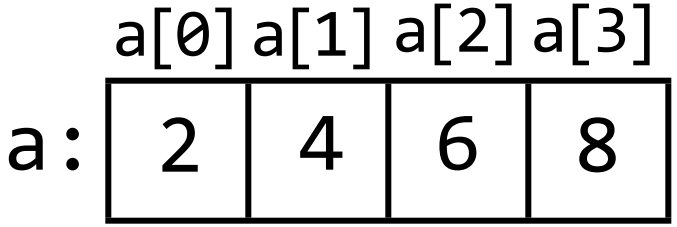
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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



2 4 6 8

Traversing Arrays

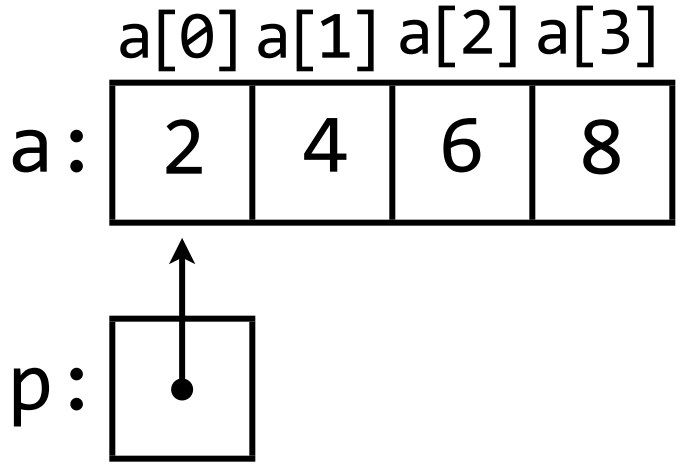
```
#define N 4
...
int a[N] = {2, 4, 6, 8};


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




```
2 4 6 8
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```



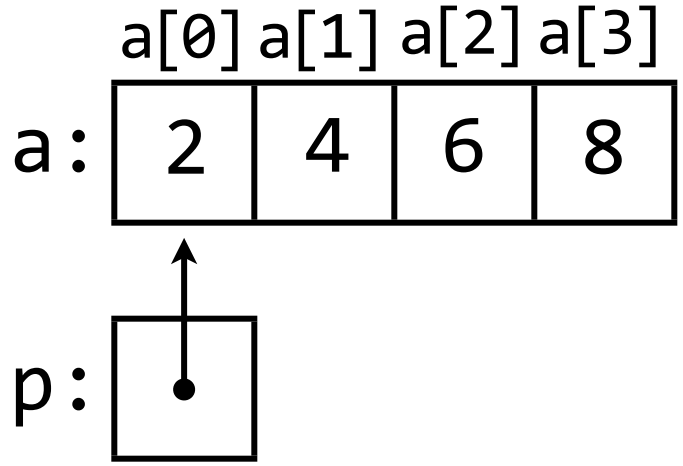

```
for (int i = 0; i < N; i++)
{
  printf("%d ", a[i]);
}
```

```
2 4 6 8
```

```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```




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

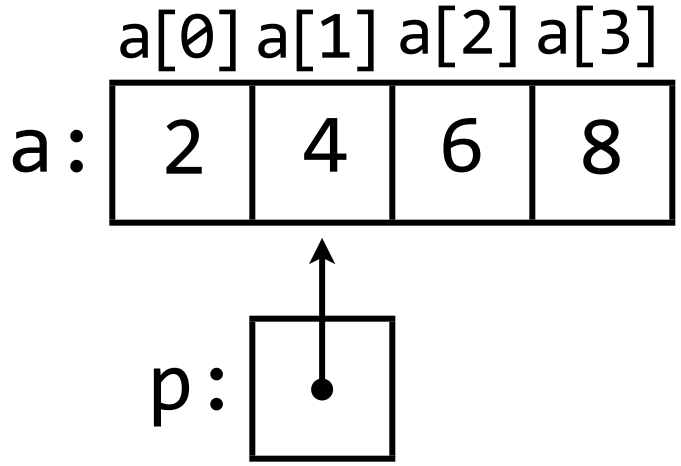
```
2 4 6 8
```

```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```




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

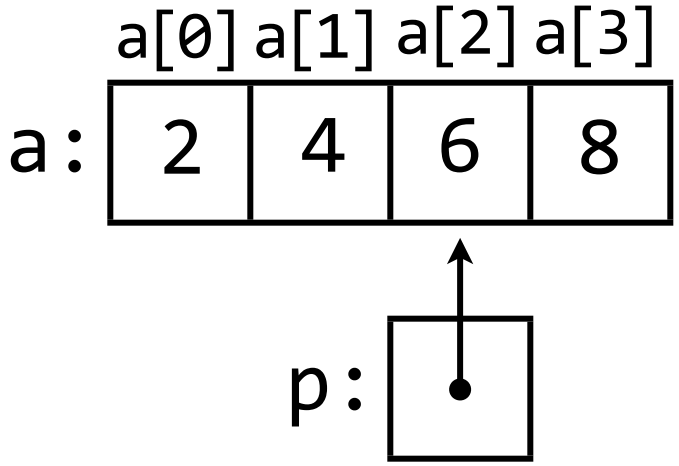
```
2 4 6 8
```

```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2 4
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```




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

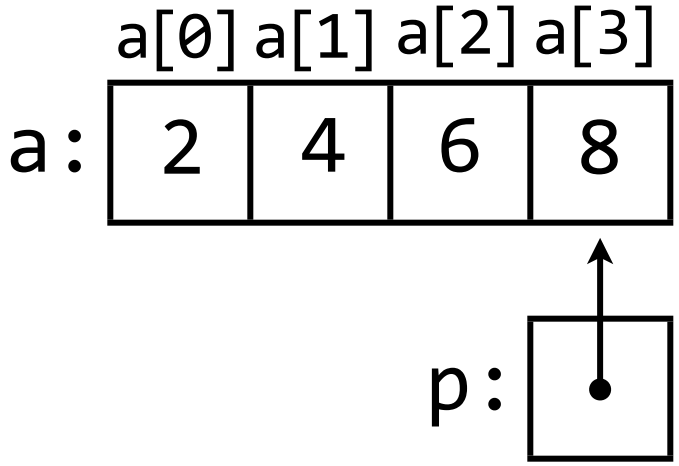
```
2 4 6 8
```

```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2 4 6
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```




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

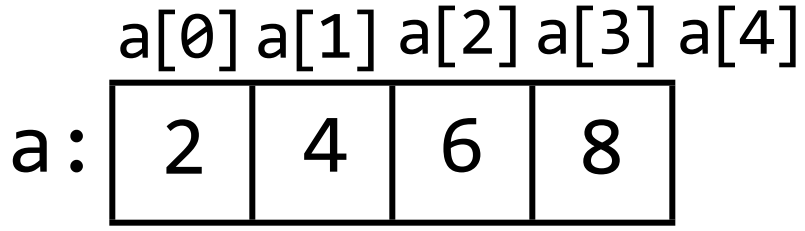
```
2 4 6 8
```

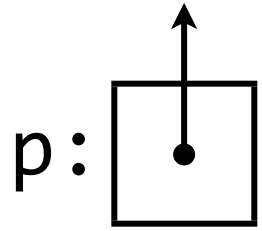
```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2 4 6 8
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```






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

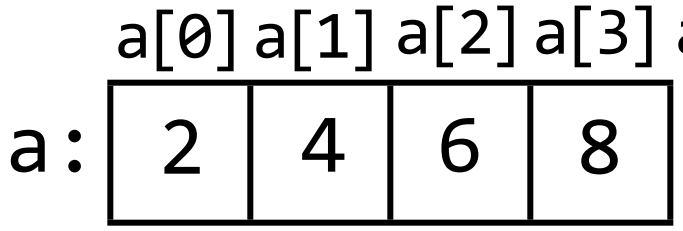
```
2 4 6 8
```

```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2 4 6 8
```

Traversing Arrays

```
#define N 4
...
int a[N] = {2, 4, 6, 8};
```




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

```
2 4 6 8
```

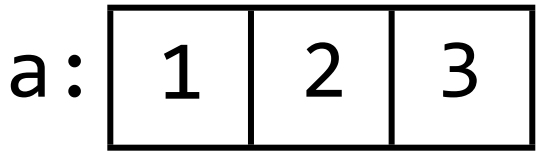
```
for (int *p = &a[0]; p < &a[N]; p++)
{
  printf("%d ", *p);
}
```

```
2 4 6 8
```

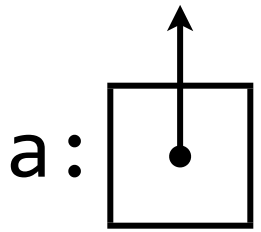
Array Names

- The name of an array can be used as a pointer to its first element

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




```
a[0] = 5;
*a = 5;
```

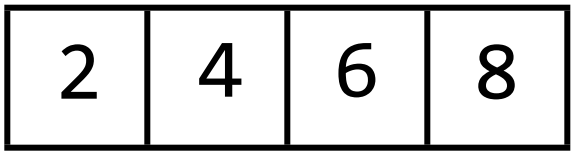


`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

```
int a[N] = {2, 4, 6, 8};
```

a:



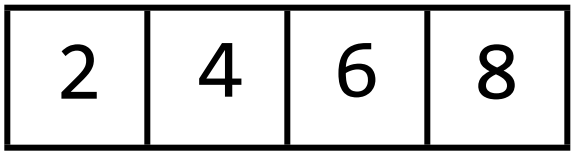

```
for (int *p = &a[0]; p < &a[N]; p++)
{
   printf("%d \n", *p);
}
```

`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

```
int a[N] = {2, 4, 6, 8};
```

a:



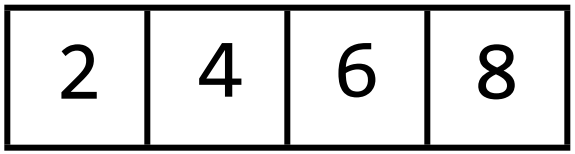

```
for (int *p = a + 0; p < &a[N]; p++)
{
   printf("%d \n", *p);
}
```

`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

```
int a[N] = {2, 4, 6, 8};
```

a:




```
for (int *p = a; p < &a[N]; p++)
{
   printf("%d \n", *p);
}
```

`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

```
int a[N] = {2, 4, 6, 8};
```

a:

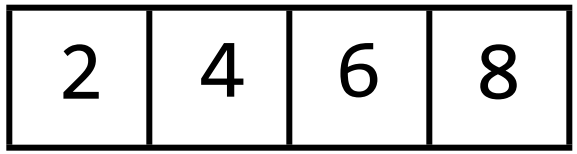
2	4	6	8

```
for (int *p = a; p < a + N; p++)
{
    printf("%d \n", *p);
}
```

`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

```
int a[N] = {2, 4, 6, 8};
```




```
for (int *p = a; p < a + N; p++)
{
    scanf("%d", p);
}
```

`*(a + i)` is the same as `a[i]`

`(a + i)` is the same as `&a[i]`

Changing Array Pointers

```
int a[N] = {2, 4, 6, 8};
int *p = a;
p++;

a++;
```

Error