Announcements

assign0 due tonight

- No late submissions

Labs start this week

- Very helpful for assign1

Goals for Today

Pointer operators

Allocating memory in the heap

malloc and free

Arrays and pointer arithmetic

Pointers

In C++, from 106B/X

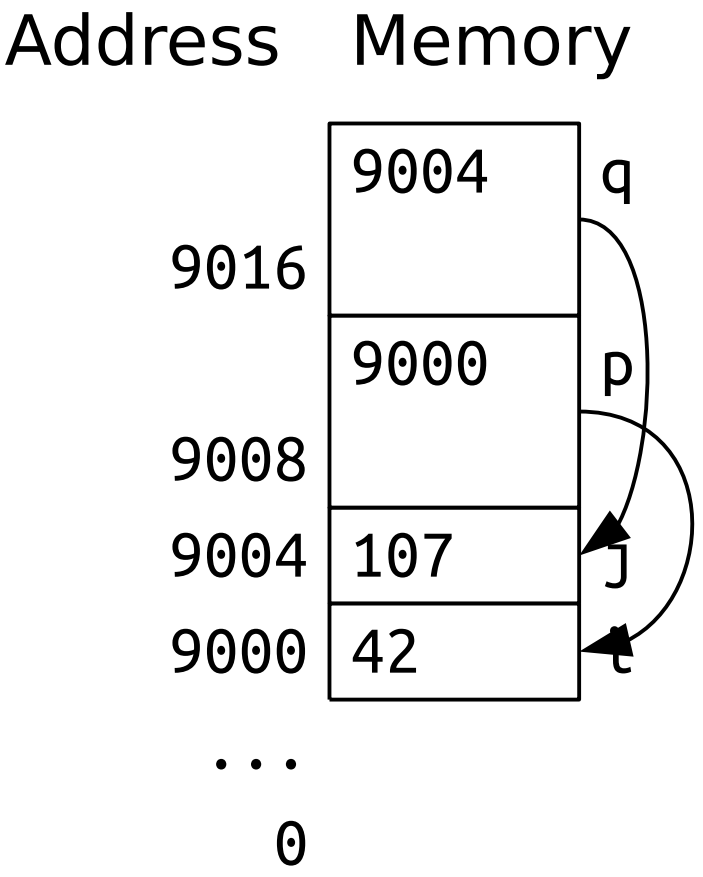
- Linked lists, trees, graphs
- Shared data

Also in C

- Arrays, strings
- Pass by reference

Pointer Operators

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;
```

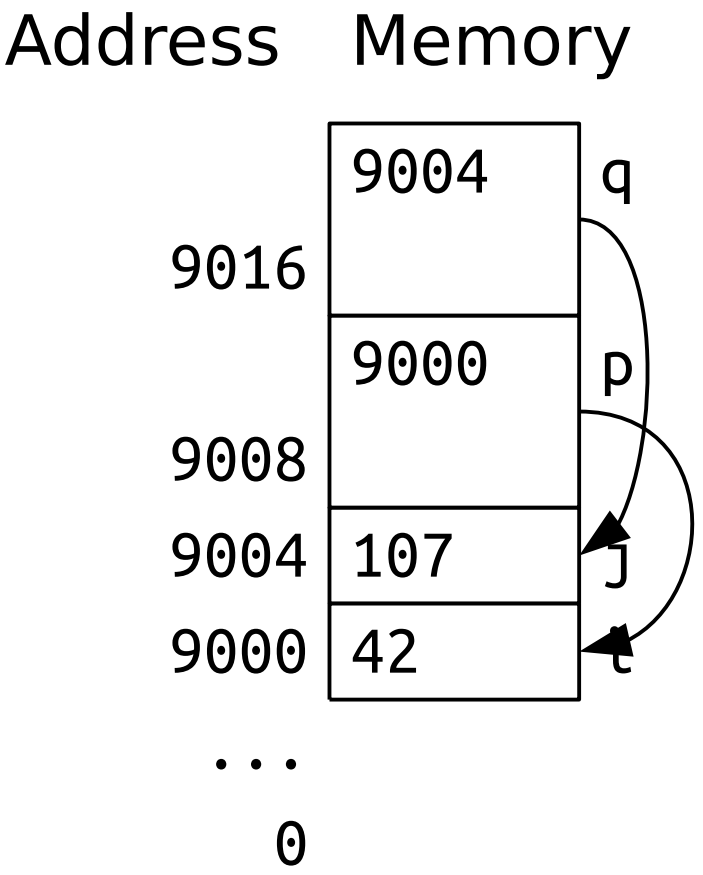


& -- address-of

& on int gives int *

Pointer Operators

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

printf("%d\n", *p); // 42
```



& -- address-of

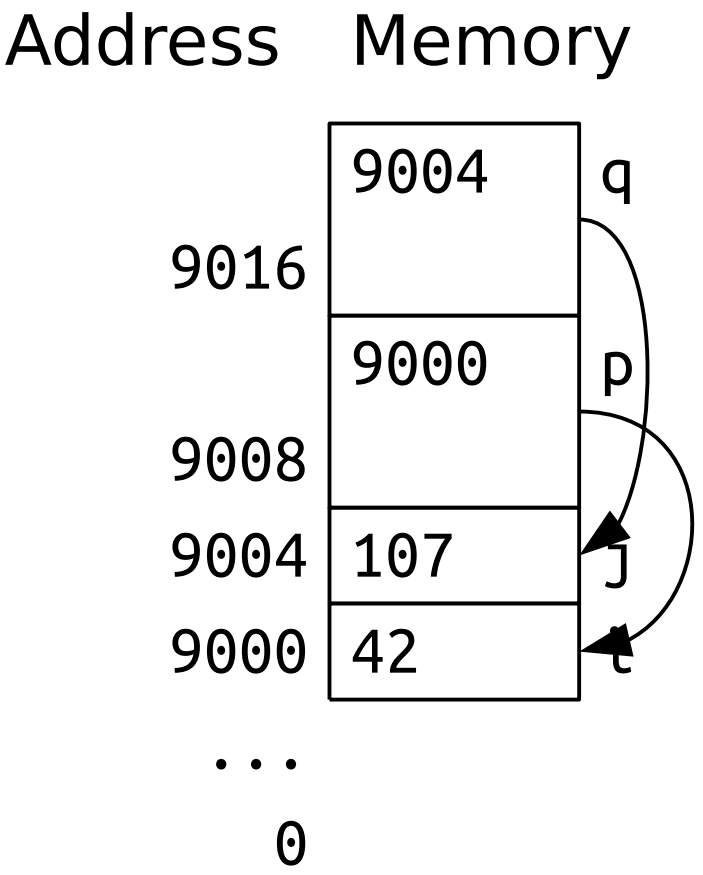
& on int gives int *

*** -- dereference**

* on int * gives int

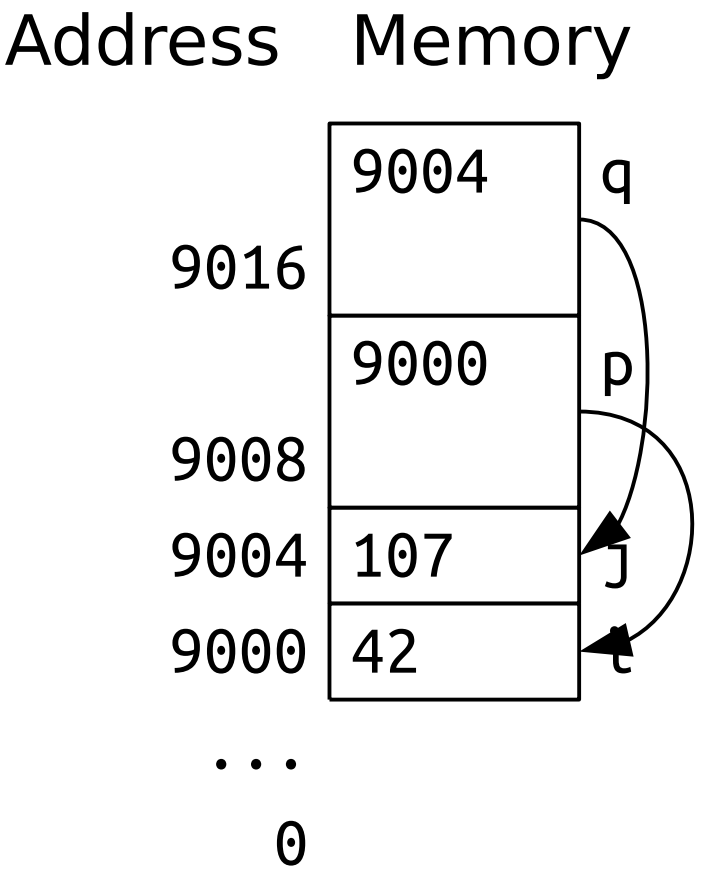
Exercise

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = q;   // (1)
*p = *q; // (2)
*p = q;  // (3)
p = *q;  // (4)
```



Exercise (1a)

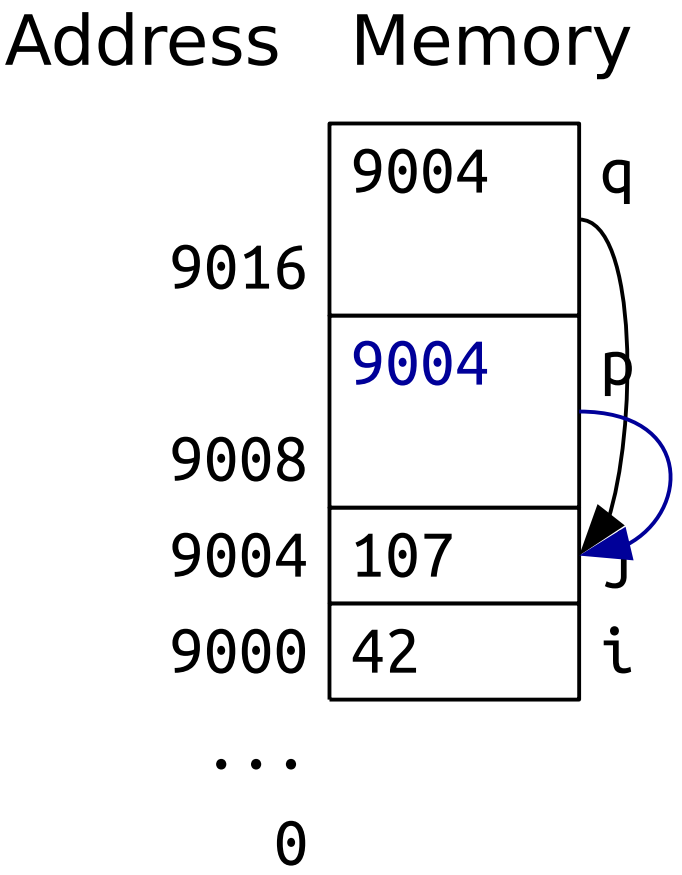
```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = q;   // (1)
```



Exercise (1b)

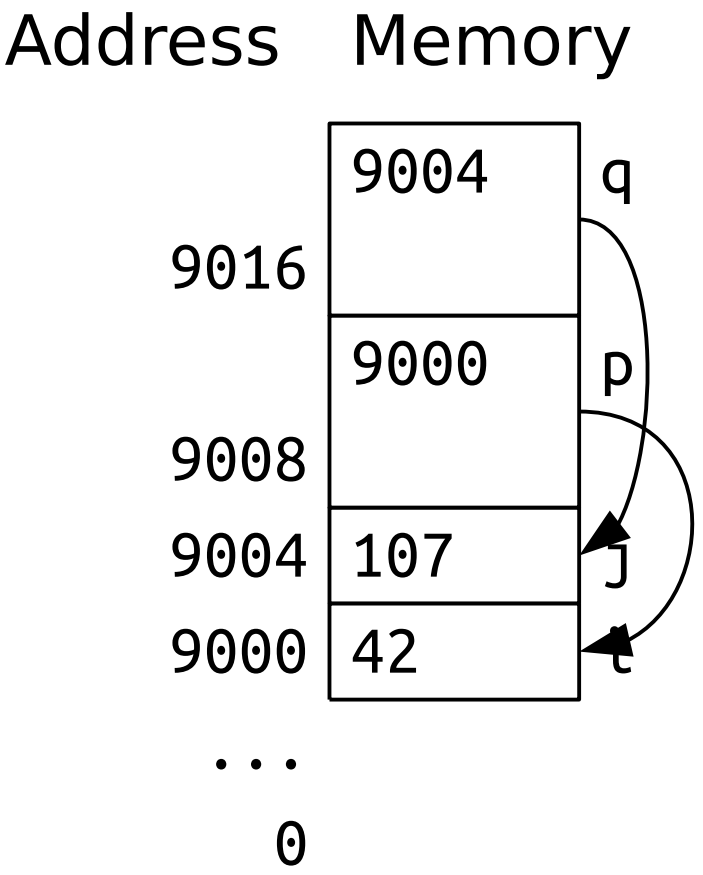
```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = q;   // (1)
```

p and q are “aliases”



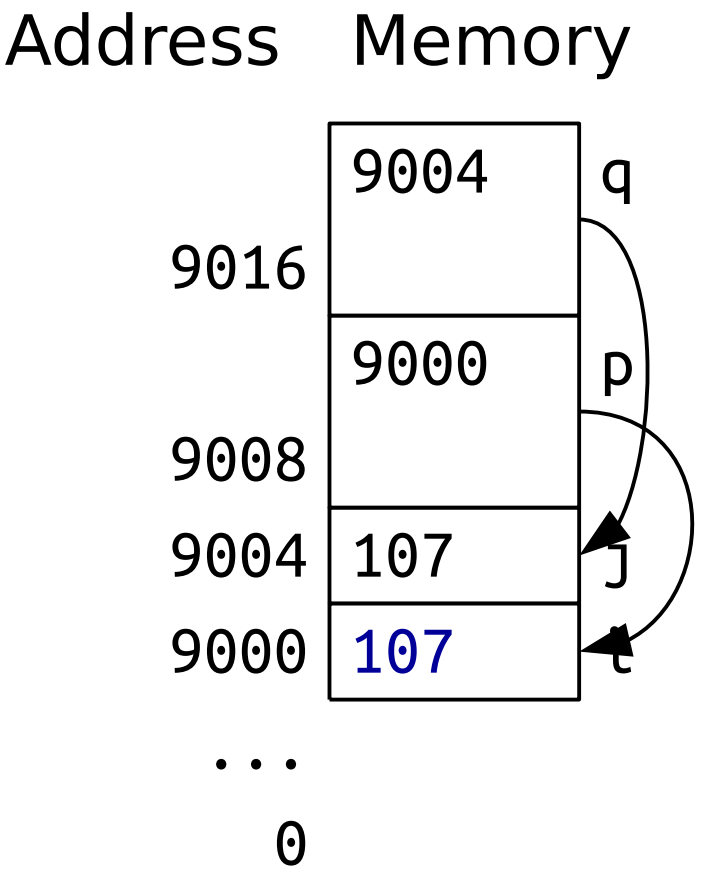
Exercise (2a)

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
*p = *q; // (2)
```



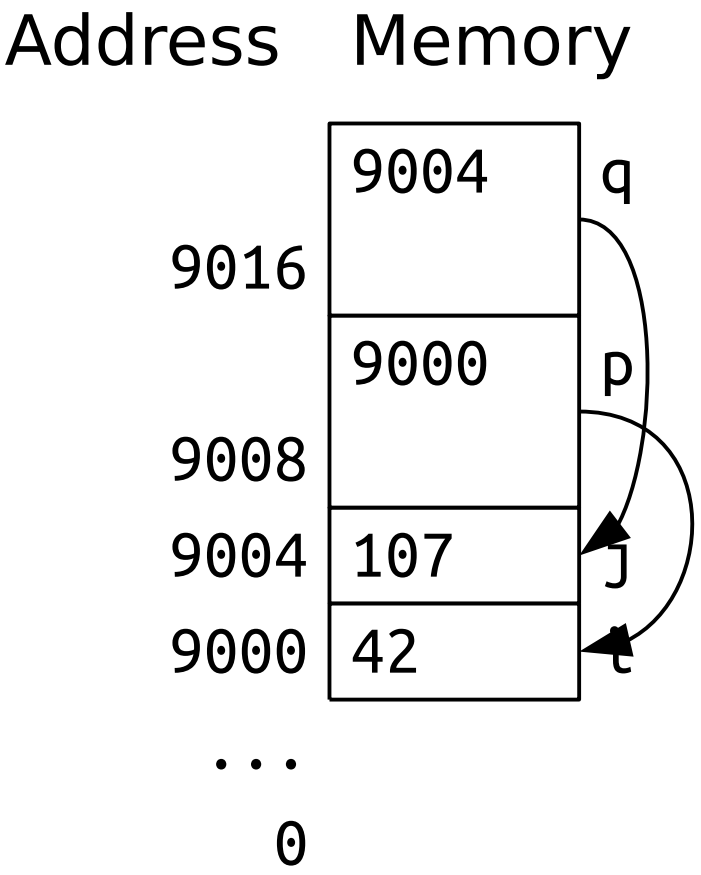
Exercise (2b)

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
*p = *q; // (2)
```



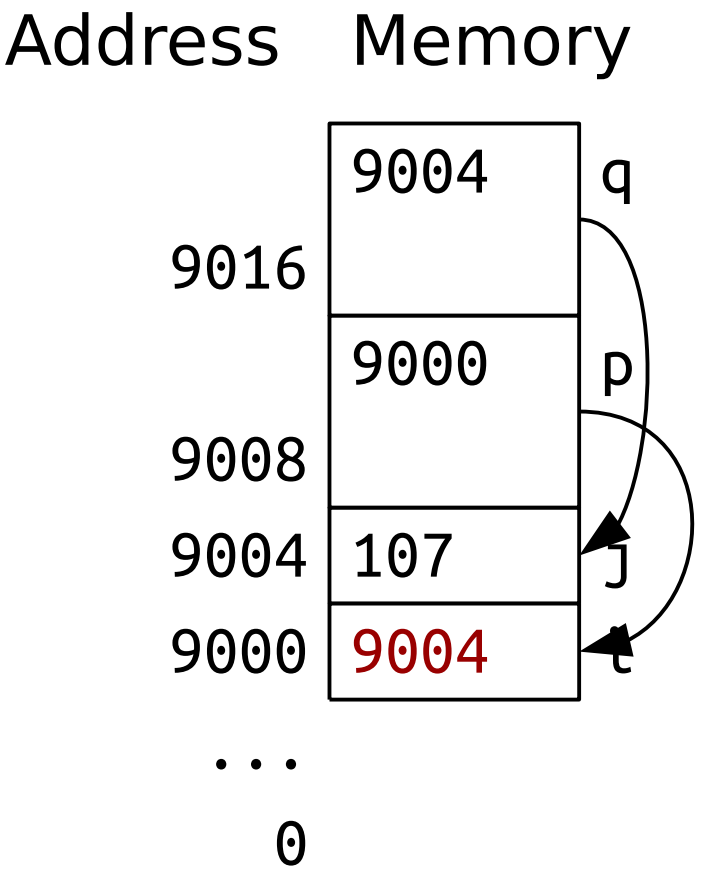
Exercise (3a)

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
*p = q;  // (3)
```



Exercise (3b)

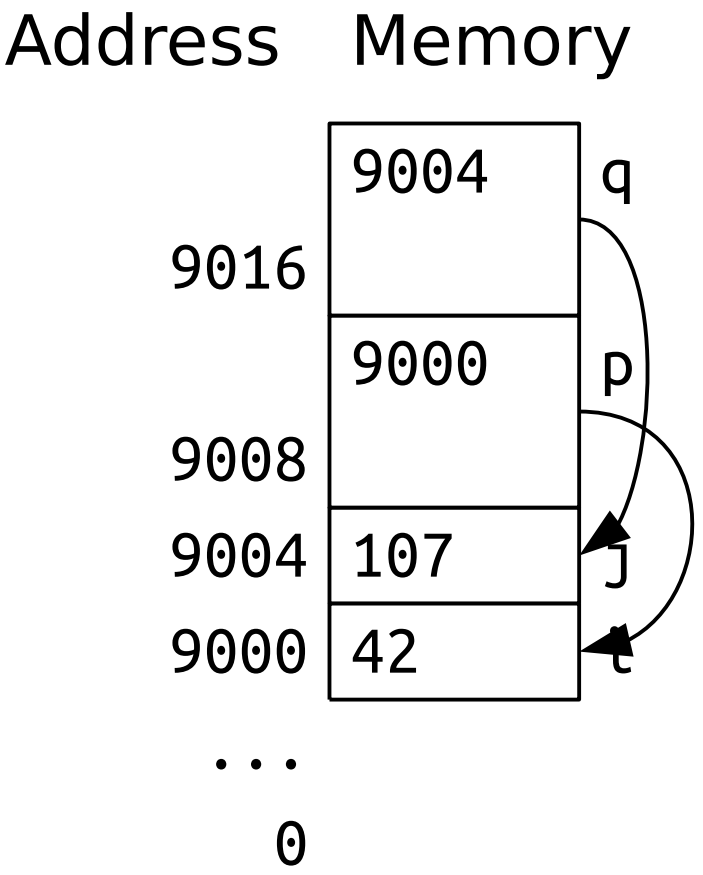
```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
*p = q;  // (3)
```



warning: assignment makes integer from pointer without a cast

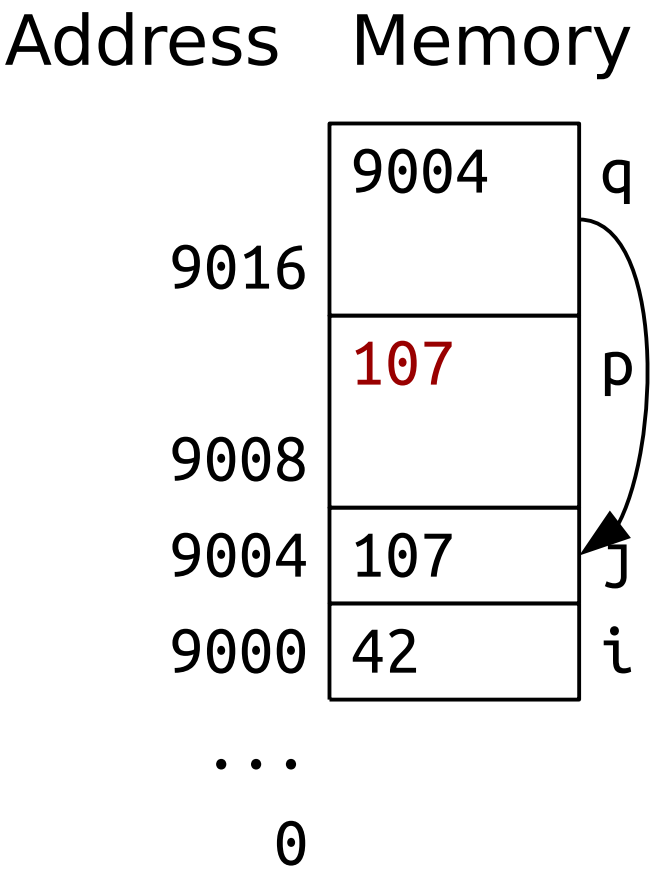
Exercise (4a)

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = *q;  // (4)
```



Exercise (4b)

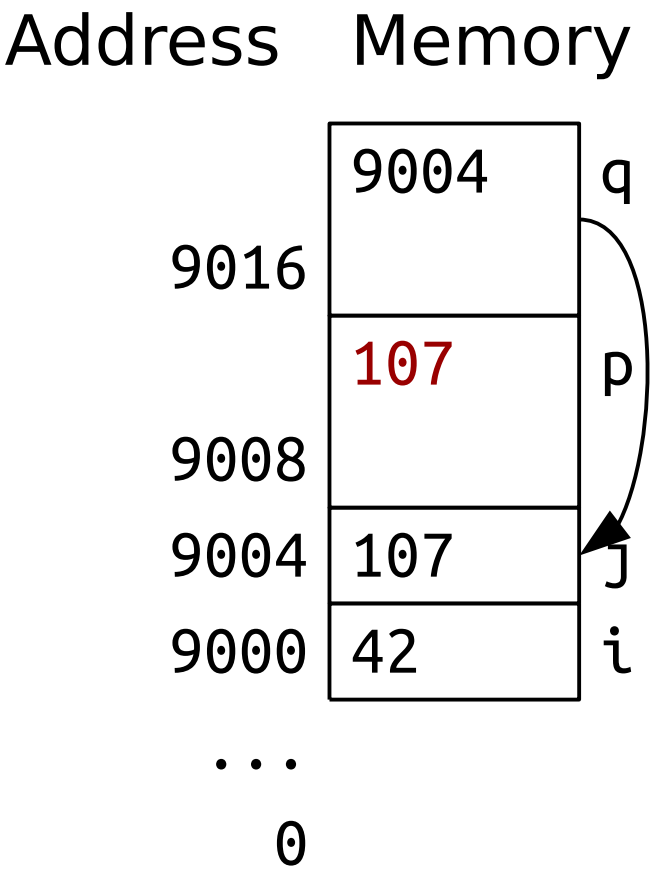
```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = *q;  // (4)
```



warning: assignment makes pointer from integer without a cast

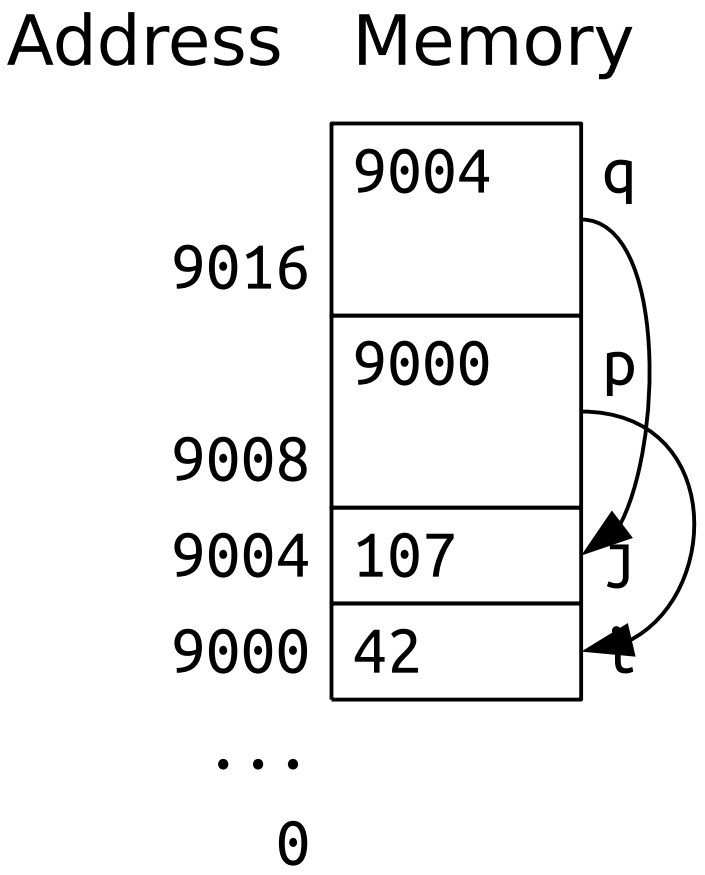
Exercise (4c)

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;

// What do these lines do?
p = *q;  // (4)

printf("%d\n", *p); // seg fault
```



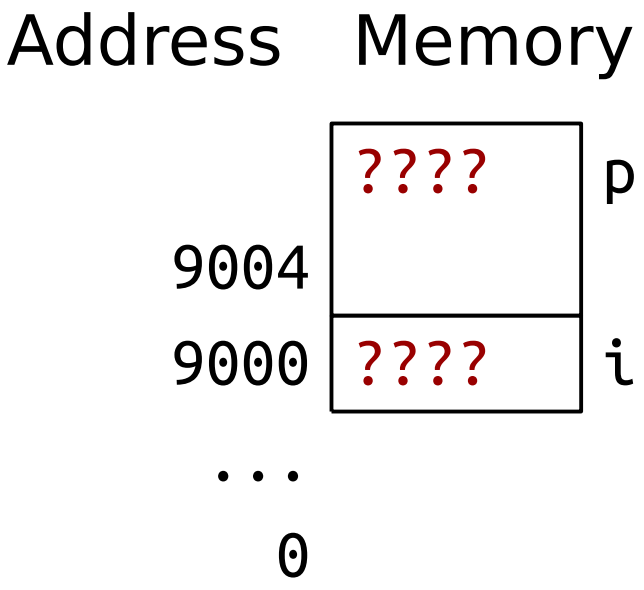
Exercise

```
int i = 42, j = 107;
int *p = &i;
int *q = &j;


p = q;   // (1)
*p = *q; // (2)
*p = q;  // (3)
p = *q;  // (4)
```



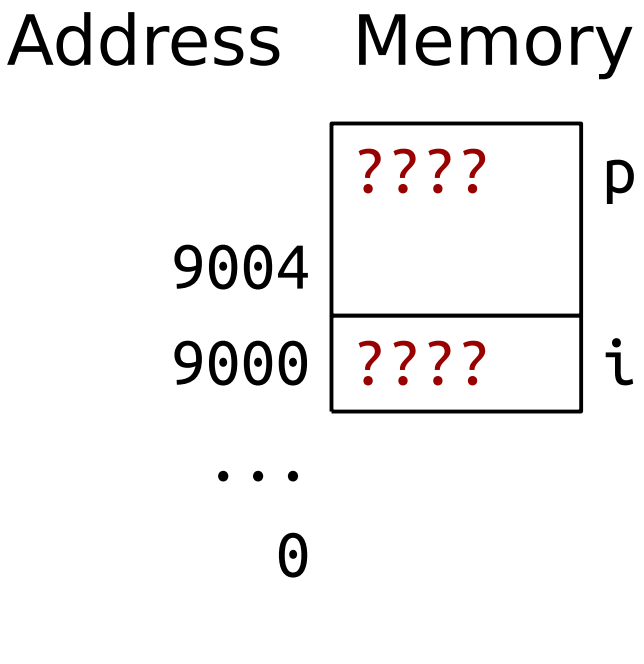
Uninitialized Pointers

```
int i;
int *p;

// “How bad” are these?
printf(“%d\n”, i);


printf(“%d\n”, *p);
```



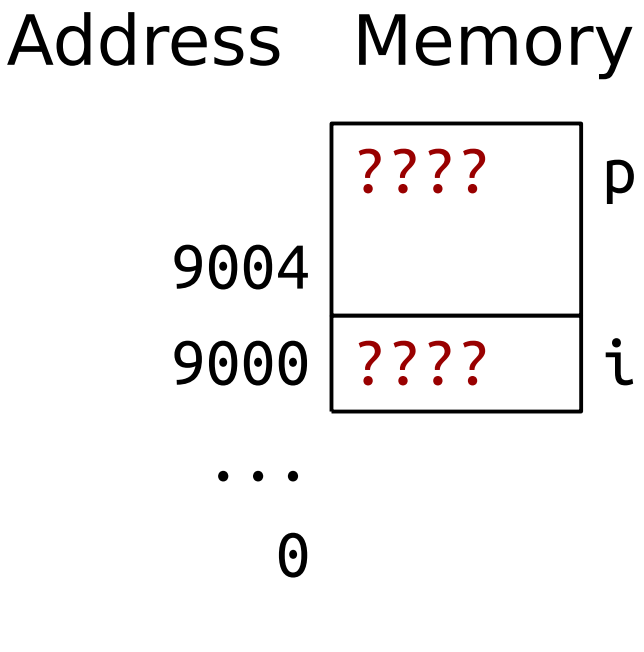
Uninitialized Pointers

```
int i;
int *p;

// “How bad” are these?
printf(“%d\n”, i);
// unpredictable, no crash

printf(“%d\n”, *p);
```



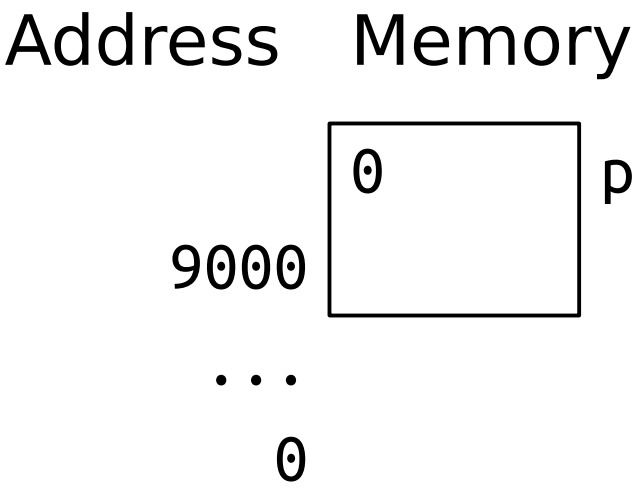
Uninitialized Pointers

```
int i;
int *p;

// “How bad” are these?
printf(“%d\n”, i);
// unpredictable, no crash

printf(“%d\n”, *p);
// probably crash
```



NULL

```
int *p = NULL;

printf(“%d\n”, *p);
// seg fault

if (p == NULL) {
  // ok
}
```



Pointee Types

```
int i = 5;
double d = 2.5;

int *ip = &i;
double *dp = &d;

dp = ip;
```

warning: assignment
from incompatible
pointer type

Allocating Memory

So far: the stack

- Lasts until function returns
- Automatically cleaned up

Now: the heap

- Recall “new” from C++

malloc

`malloc(size)`

- Argument is number of bytes needed
- Returns pointer to space i nheap
- Memory stays around until freed

malloc

`malloc(size)`

- Argument is number of bytes needed
- Returns pointer to space i nheap
- Memory stays around until freed

`sizeof(type)`

- Number of bytes a given type needs

malloc and free

`malloc(size)`

- Argument is number of bytes needed
- Returns pointer to space i nheap
- Memory stays around until freed

`sizeof(type)`

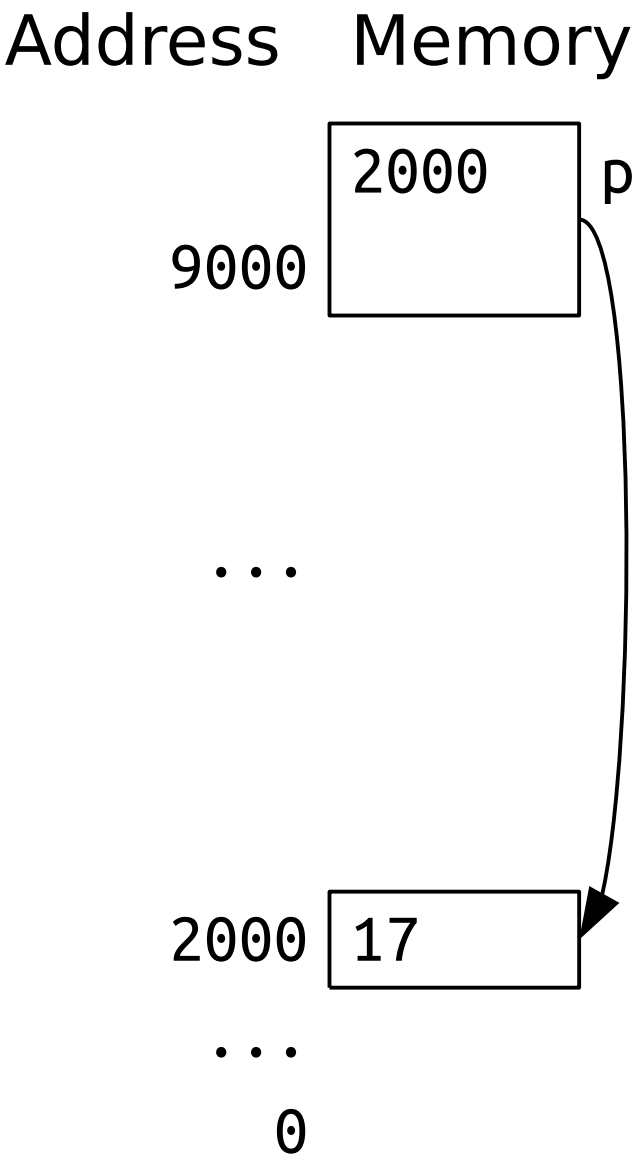
- Number of bytes a given type needs

`free(ptr)`

- Mark memory as no longer in use

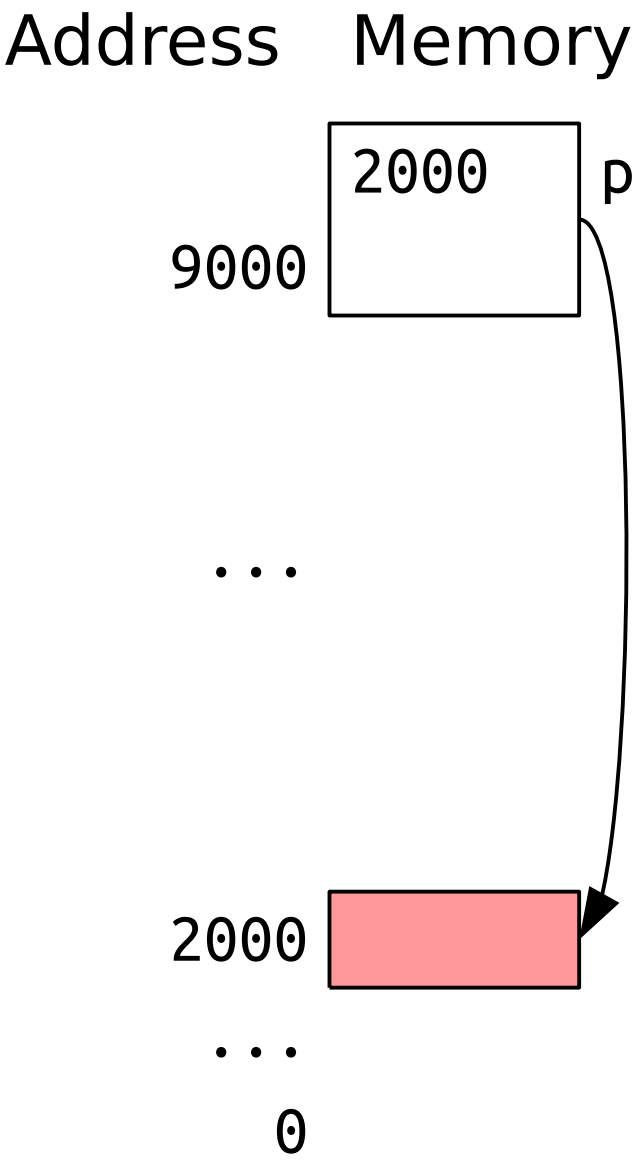
Using malloc and free

```
int *p = malloc(sizeof(int));
*p = 17;
```



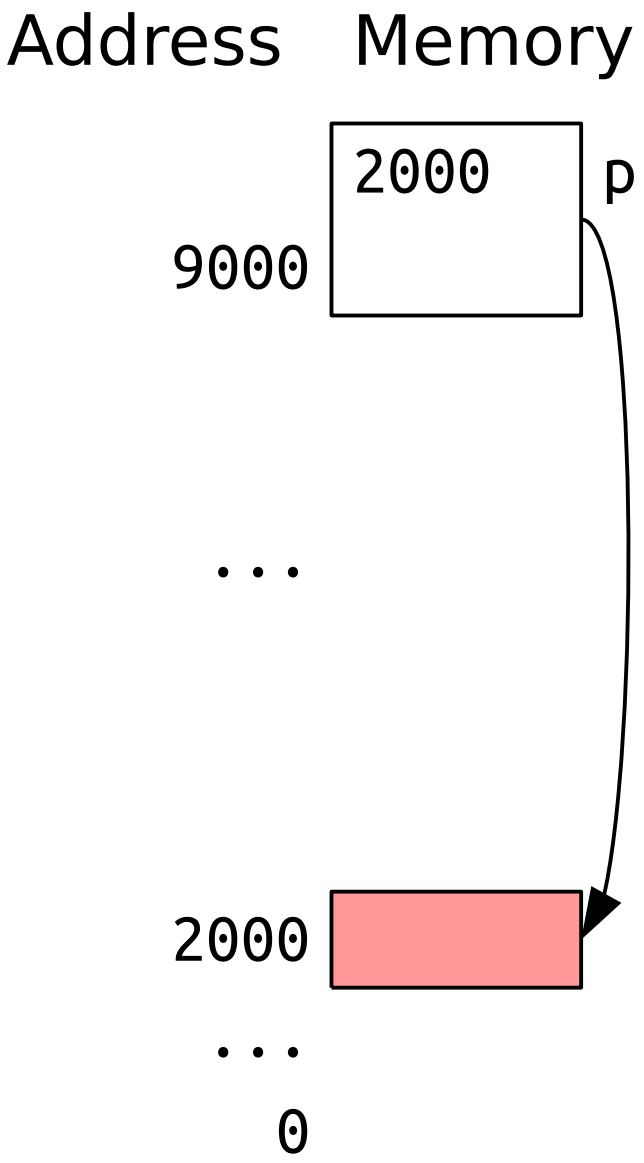
Using malloc and free

```
int *p = malloc(sizeof(int));
*p = 17;

free(p);
```



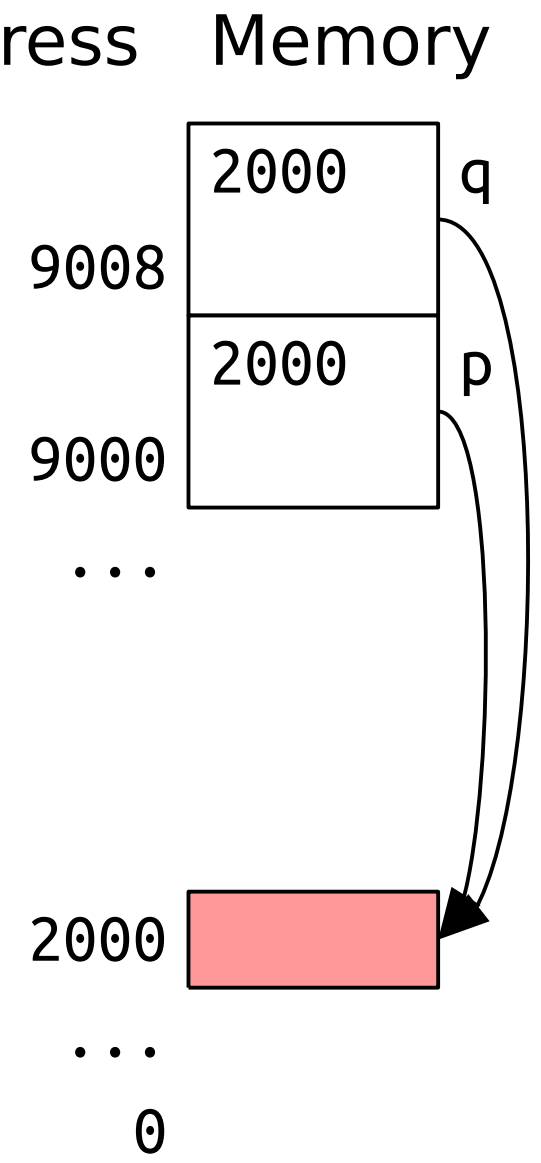
Using malloc and free

```
int *p = malloc(sizeof(int));
*p = 17;

free(p);

// BUGGY
printf("%d\n", *p);
*p = 35;
free(p);
```



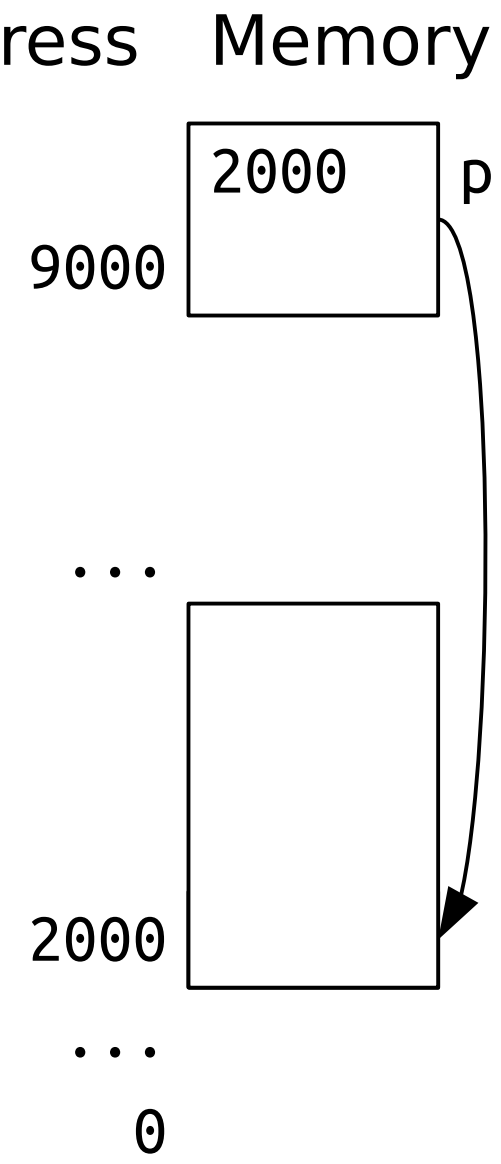
Using malloc and free

```
int *p = malloc(sizeof(int));
*p = 17;
int *q = p;
free(p);

free(q); // BUGGY
```



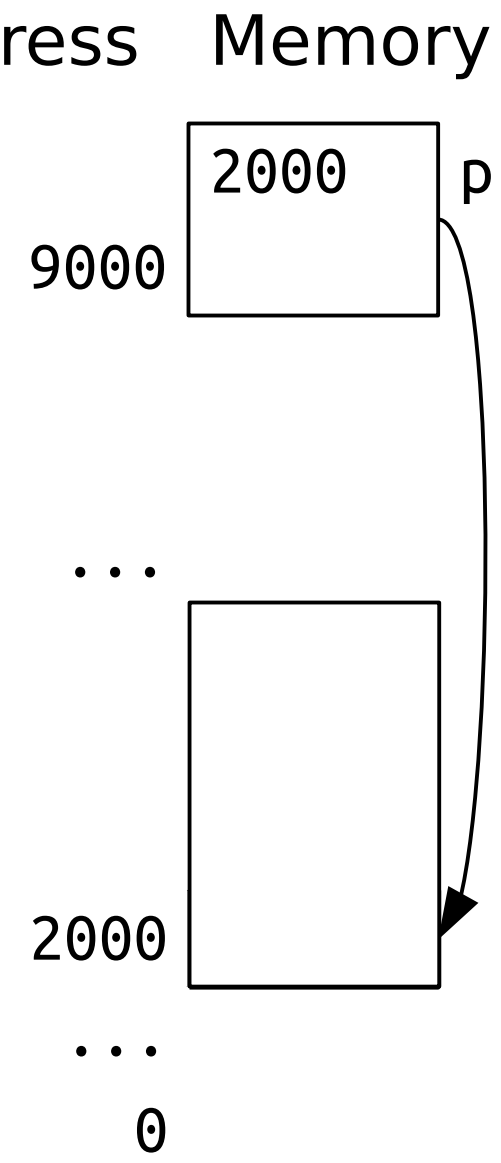
Arrays?

```
int *p = malloc(4 * sizeof(int));
```



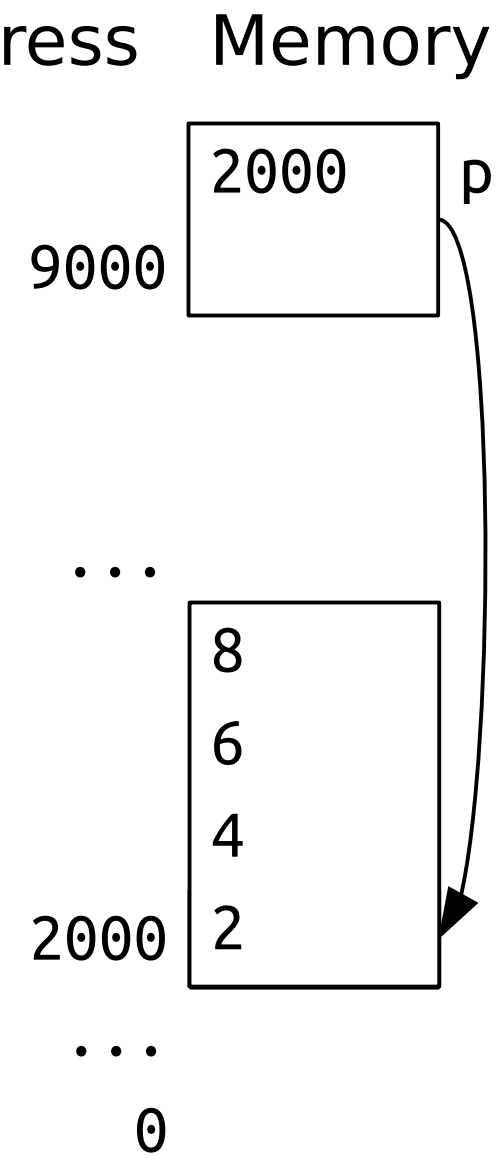
Arrays?

```
int *p = malloc(4 * sizeof(int));
// What if we could do...
p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;
```



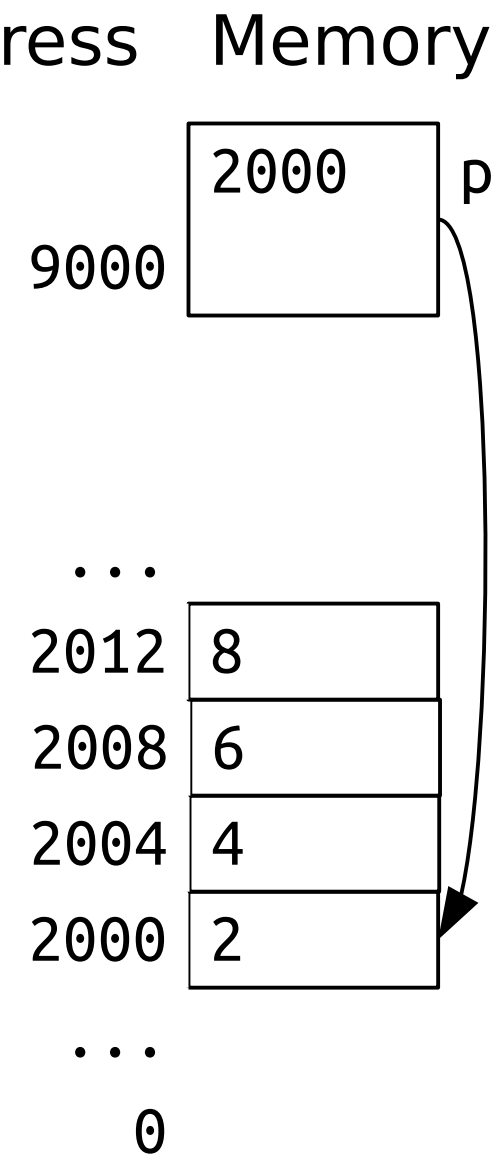
Arrays!

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;
```



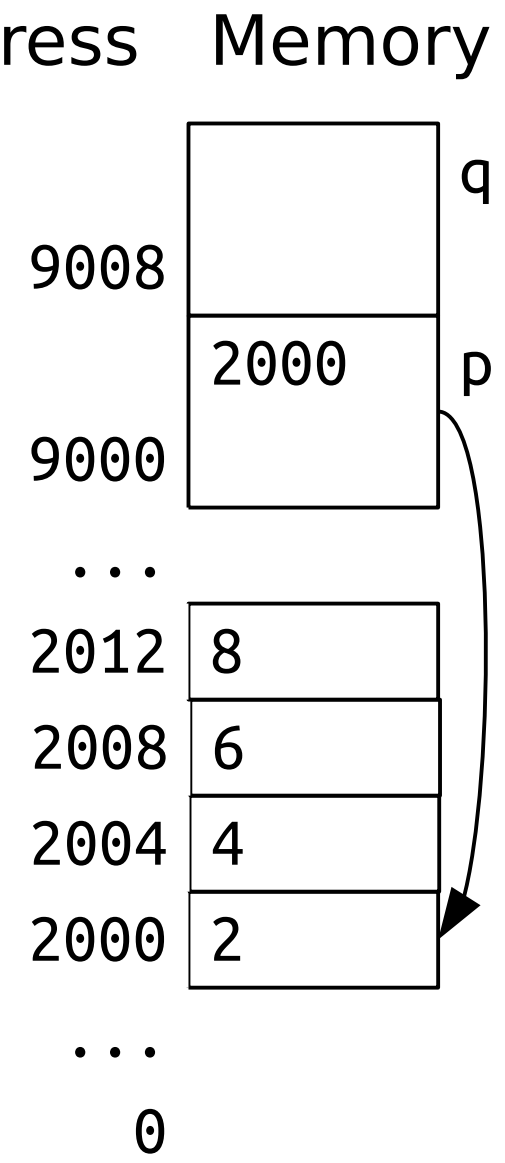
Arrays

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;
```



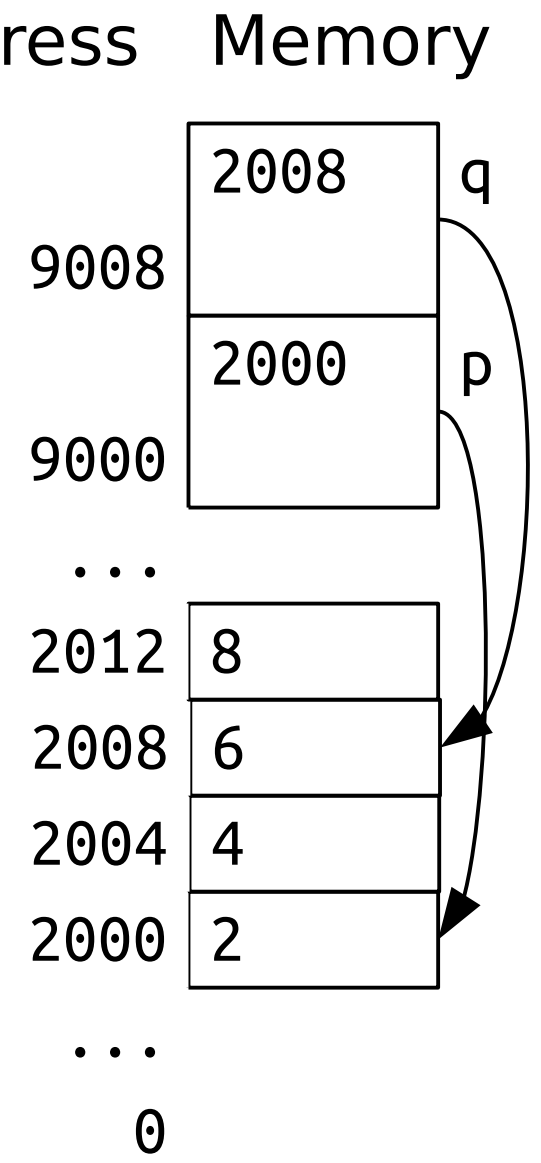
Pointer Arithmetic

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
```



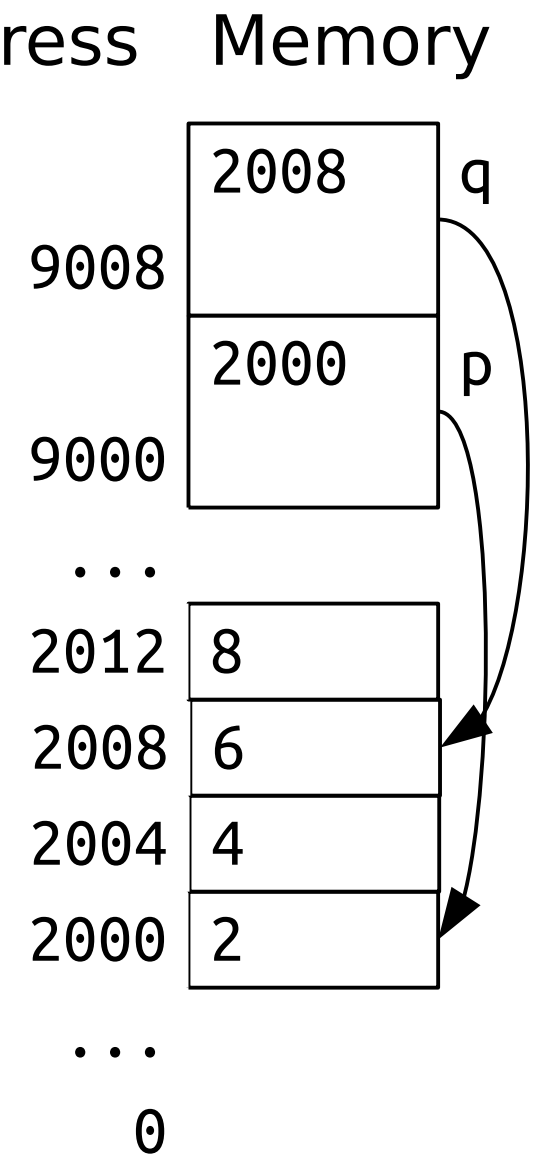
Pointer Arithmetic

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
printf("%d\n", *q); // 6
```



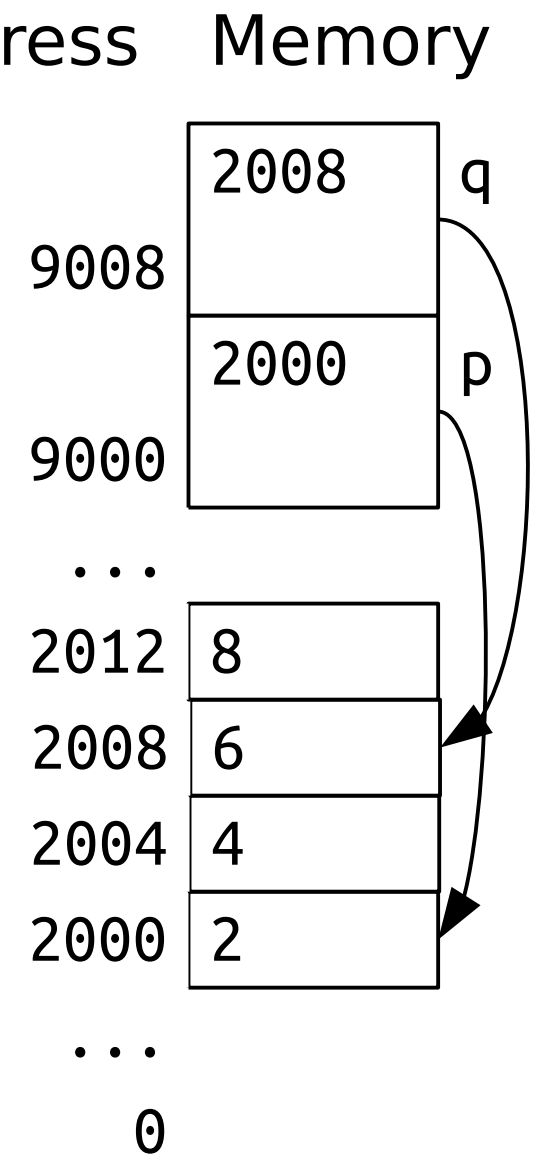
Pointer Arithmetic

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
printf("%d\n", *q); // 6

// p[2] == *(p + 2)
```



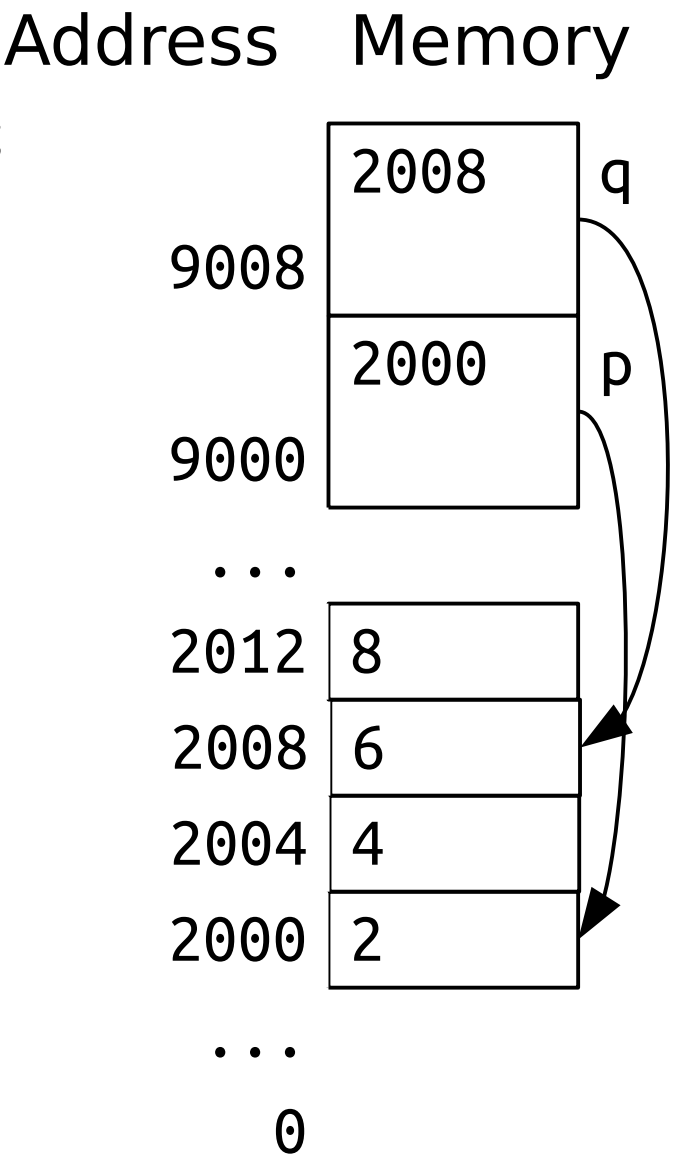
Pointer Arithmetic

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
printf("%d\n", *q); // 6

// p[2] == *(p + 2)
// p[0] == *(p + 0) == *p
```



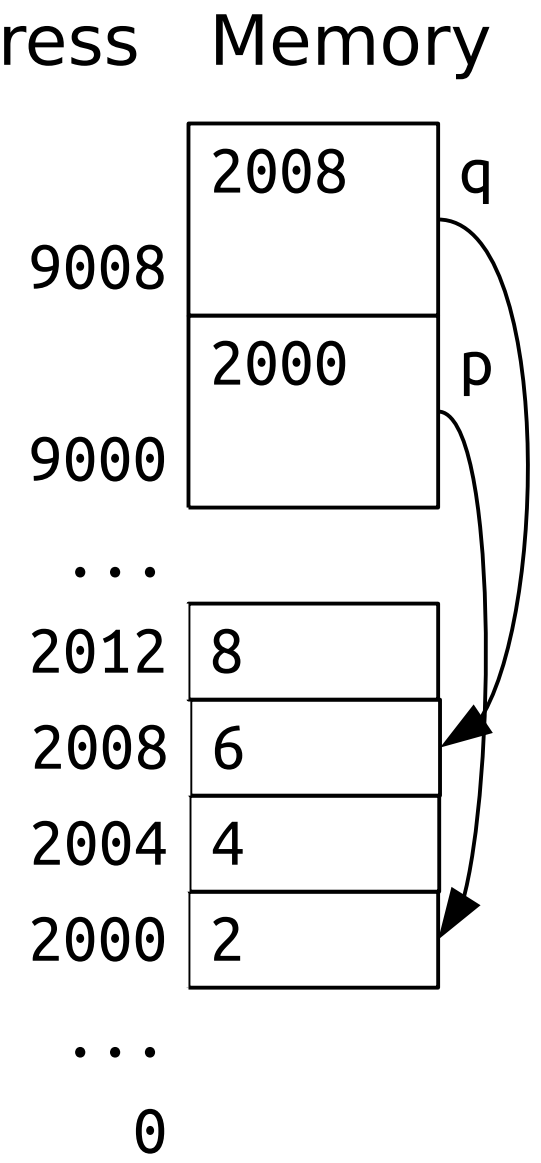
Pointer Arithmetic

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
printf("%d\n", *q); // 6

// p[2] == *(p + 2)
// p[0] == *(p + 0) == *p
// &p[2] == p + 2
// &p[0] == p
```



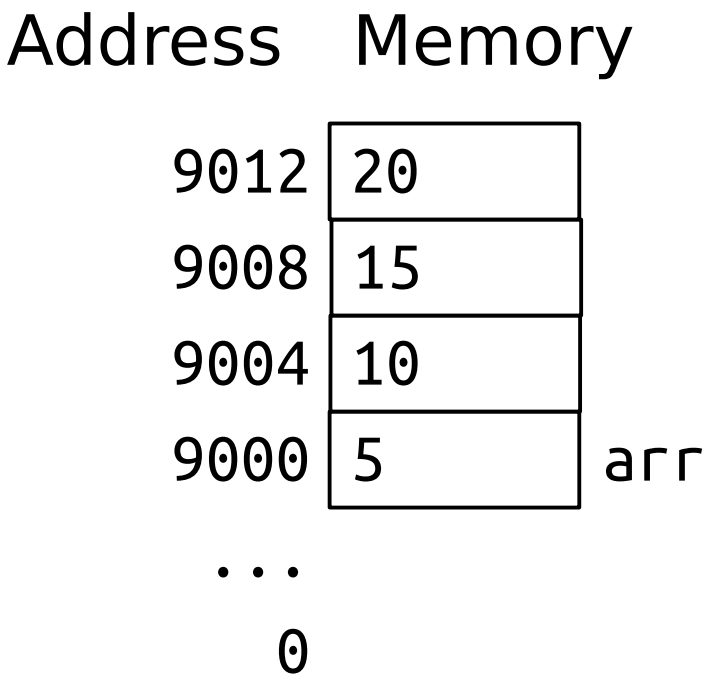
Subarrays

```
int *p = malloc(4 * sizeof(int));

p[0] = 2; p[1] = 4;
p[2] = 6; p[3] = 8;

int *q = p + 2;
printf("%d\n", q[1]); // 8
```



Stack Arrays

```
int arr[4];
arr[0] = 5;  arr[1] = 10;
arr[2] = 15; arr[3] = 20;

// arr == &arr[0]
```



Strings

Recall char * from Friday

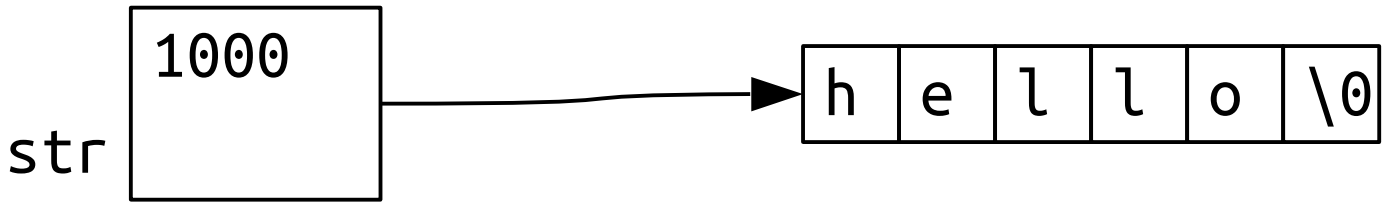
Called “C string”

Pointer to character(s)

How does printf know when to stop printing?

‘\0’ (null terminator) at end

```
char *str = “hello”;
```



Summary

Pointer operators

Allocating memory in the heap

malloc and free

Arrays and pointer arithmetic

Lab: Strings

Next time: More pointer uses

Pass by reference, stack vs. heap