

# CSE 333

## Lecture 3 - pointers, pointers, pointers

**Hal Perkins**

Department of Computer Science & Engineering

University of Washington



# Administrivia 1

hw0 out, due Tuesday night

- Infrastructure shakedown cruise. Basic assignment should be pretty simple
  - ▶ clint and valgrind tools will be useful for exercises also
- Need to get this into your CSE gitlab repo. Use discussion board to sort out issues; mail to [cse333-staff@cs](mailto:cse333-staff@cs) as needed

ex2 out now, due Monday before class

# Administrivia 2

## Exercise grading

- Still a work in progress
- Some things to watch for:
  - ▶ No input sanity check (i.e., there should be an argument, it should be an int)
  - ▶ Single blob of code, no functions
  - ▶ Formatting funnies (tabs, not spaces) - likely just a warning this time
    - Suggestion: now that you have `clint.py`, use it to check exercises
- Coding question: why compute  $x^y$  to alternate `+-+--+-...`?

# Administrivia 3

## Documentation vs Folklore

- Documentation: man pages; books; reference web sites like [cplusplus.com](http://cplusplus.com), [man7.org](http://man7.org); ...
- Folklore: google, stackoverflow, the person down the hall

## Tradeoffs, relative strengths/weaknesses

- Discuss

# Agenda

## Today's goals:

- pointers
- more pointers
- pointers and call-by-reference
- arrays and pointers

# & and \*

*&foo* // virtual address of foo - “address of”  
*\*pointer* // dereference a pointer  
*\*pointer = value;* // dereference / assign

deref.c

```
int x = 42;  
int *p; // p is a pointer to an integer  
p = &x; // p now stores the address of x  
  
printf("x is %d\n", x);  
*p = 99;  
printf("x is %d\n", x);
```

# Something curious

Let's try running this program several times:

asr.c

```
#include <stdio.h>

int main(int argc, char **argv) {
    int x = 1;
    int *p = &x;

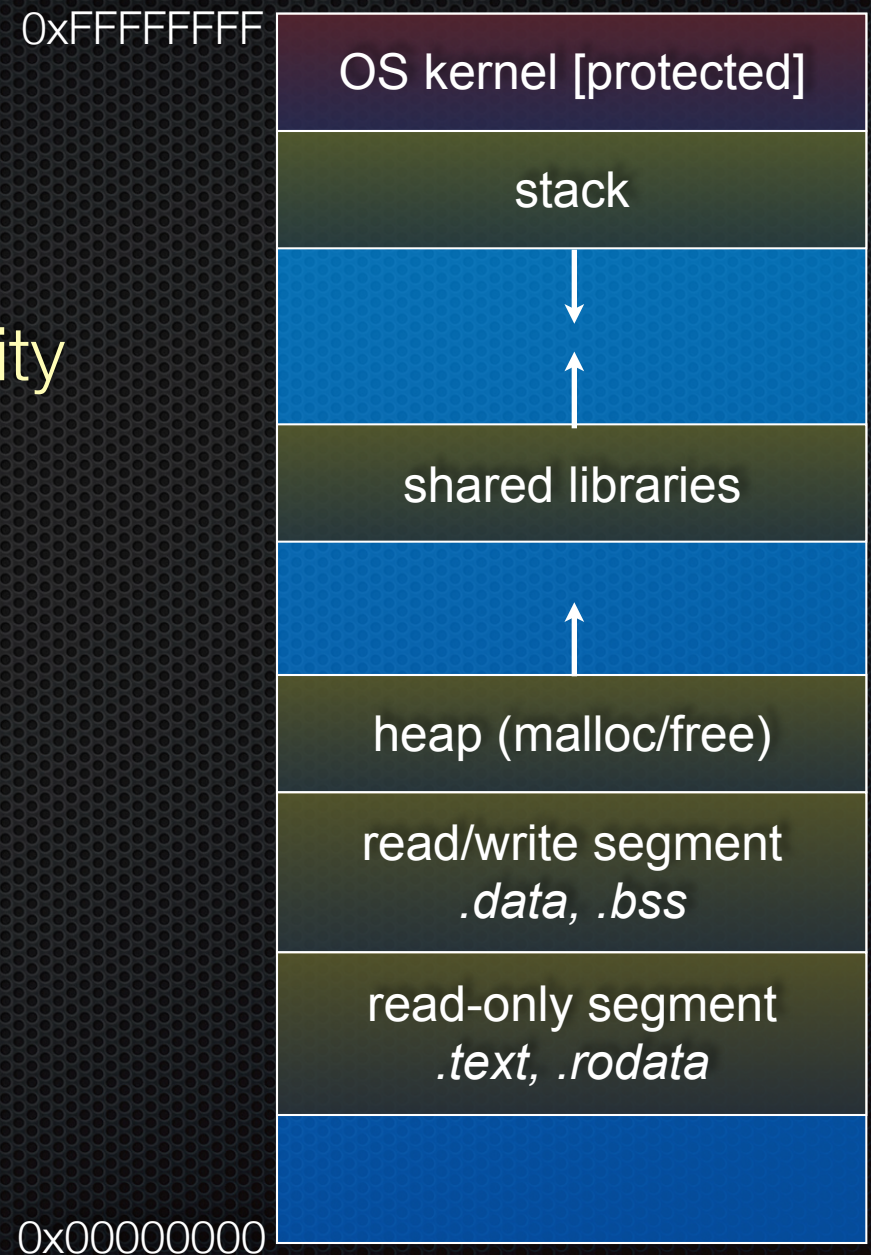
    printf("&x: %p; p: %p; &p: %p\n",
           &x, p, &p);
    return 0;
}
```

```
[bash]$ ./asr
&x: 0xbfa521dc; p: 0xbfa521dc; &p: 0xbfa521d8
[bash]$ ./asr
&x: 0xbf836f5c; p: 0xbf836f5c; &p: 0xbf836f58
[bash]$ ./asr
&x: 0xbfea39dc; p: 0xbfea39dc; &p: 0xbfea39d8
```

# ASR

Linux uses address-space randomization for added security

- linux randomizes:
  - ▶ base of stack
  - ▶ shared library (mmap) location
- makes stack-based buffer overflow attacks tougher
- makes debugging tougher
- google “disable linux address space randomization”





# Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

address

name	value
------	-------

&x

x	value
---	-------

&arr[0]

arr[0]	value
--------	-------

&arr[1]

arr[1]	value
--------	-------

&arr[2]

arr[2]	value
--------	-------

&p

p	value
---	-------

# Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

address

name	value
------	-------

&x

x	1
---	---

&arr[0]

arr[0]	2
--------	---

&arr[1]

arr[1]	3
--------	---

&arr[2]

arr[2]	4
--------	---

&p

p	&arr[1]
---	---------

# Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

address

name	value
------	-------

0xbfff2dc

<b>x</b>	1
----------	---

0xbfff2d0

<b>arr[0]</b>	2
---------------	---

0xbfff2d4

<b>arr[1]</b>	3
---------------	---

0xbfff2d8

<b>arr[2]</b>	4
---------------	---

0xbfff2cc

<b>p</b>	0xbfff2d4
----------	-----------

# Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

address

name	value
------	-------

0xbfff2dc

**x**

1

0xbfff2d8

**arr[2]**

4

0xbfff2d4

**arr[1]**

3

0xbfff2d0

**arr[0]**

2

0xbfff2cc

**p**

0xbfff2d4

main()'s stack frame

# Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
    int x = 1;
    int arr[3] = {2, 3, 4};
    int *p = &arr[1];

    printf("&x: %p; x: %d\n", &x, x);
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

    return 0;
}
```

address

name	value
------	-------

0xbfff2dc

<b>x</b>	1
----------	---

0xbfff2d0

<b>arr[0]</b>	2
---------------	---

0xbfff2d4

<b>arr[1]</b>	3
---------------	---

0xbfff2d8

<b>arr[2]</b>	4
---------------	---

0xbfff2cc

<b>p</b>	0xbfff2d4
----------	-----------



# Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p; ←  
  
   >(*dp) += 1;  
    p += 1;  
   >(*dp) += 1;  
  
    return 0;  
}
```

address

name	value
------	-------

0xbfff2dc

x	1
---	---

0xbfff2d0

arr[0]	2
--------	---

0xbfff2d4

arr[1]	3
--------	---

0xbfff2d8

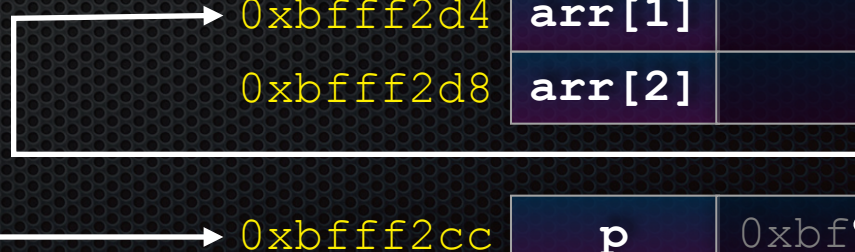
arr[2]	4
--------	---

0xbfff2c8

dp	0xbfff2cc
----	-----------

0xbfff2cc

p	0xbfff2d4
---	-----------



# Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
   >(*dp) += 1; ←  
    p += 1;  
   >(*dp) += 1;  
  
    return 0;  
}
```

address

name	value
------	-------

0xbfff2dc

x	1
---	---

0xbfff2d0

arr[0]	2
--------	---

0xbfff2d4

arr[1]	4
--------	---

0xbfff2d8

arr[2]	4
--------	---

0xbfff2c8

dp	0xbfff2cc
----	-----------

0xbfff2cc

p	0xbfff2d4
---	-----------



# Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
   >(*dp) += 1;  
    p += 1; ←  
   >(*dp) += 1;  
  
    return 0;  
}
```

address

name	value
------	-------

0xbfff2dc

x	1
---	---

0xbfff2d0

arr[0]	2
--------	---

0xbfff2d4

arr[1]	4
--------	---

0xbfff2d8

arr[2]	4
--------	---

0xbfff2c8

dp	0xbfff2cc
----	-----------

0xbfff2cc

p	0xbfff2d8
---	-----------





# Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
   >(*dp) += 1;  
    p += 1;  
   >(*dp) += 1; ←  
  
    return 0;  
}
```

address

name	value
------	-------

0xbfff2dc

<b>x</b>	1
----------	---

0xbfff2d0

<b>arr[0]</b>	2
---------------	---

0xbfff2d4

<b>arr[1]</b>	4
---------------	---

0xbfff2d8

<b>arr[2]</b>	<b>5</b>
---------------	----------

0xbfff2c8

<b>dp</b>	0xbfff2cc
-----------	-----------

0xbfff2cc

<b>p</b>	0xbfff2d8
----------	-----------



# Pointer arithmetic

## Pointers are typed

- `int *int_ptr; VS. char *char_ptr;`
- pointer arithmetic obeys those types
- *see `pointerarithmetic.c`*

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

0xFFFFFFFF

stack  
(assume 32-bit x86)

char\_ptr

int\_ptr

arr[2]

arr[1]

arr[0]

0x00000000

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

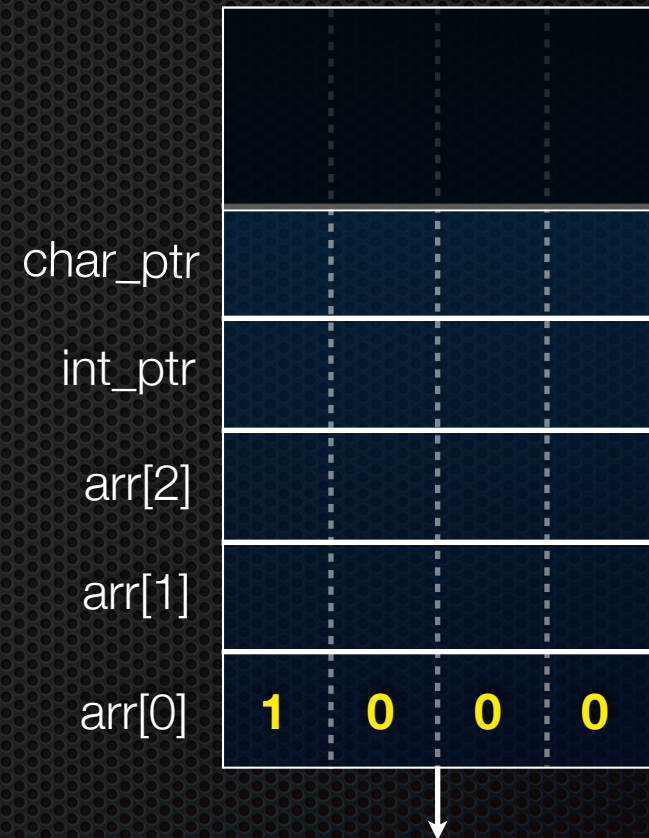
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



(x86 is little endian)

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

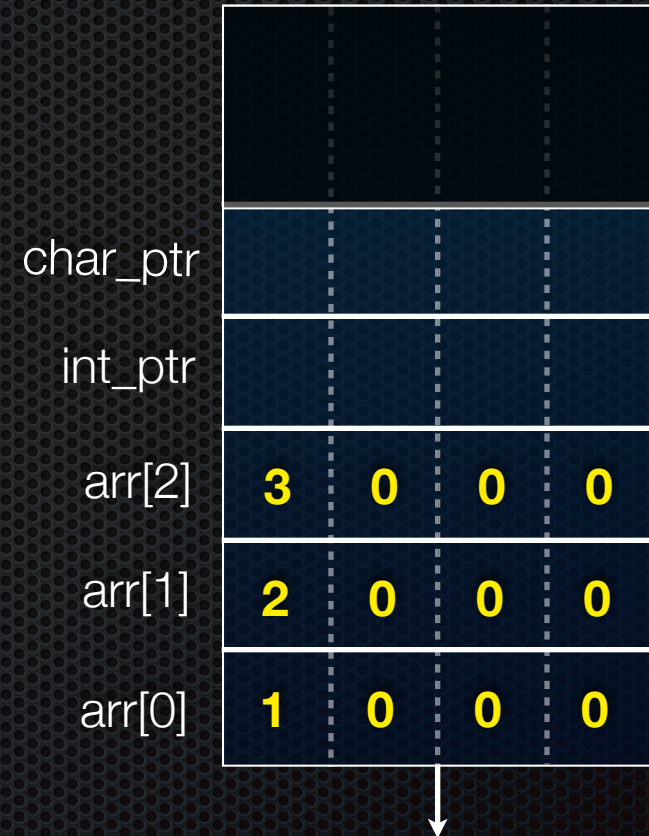
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

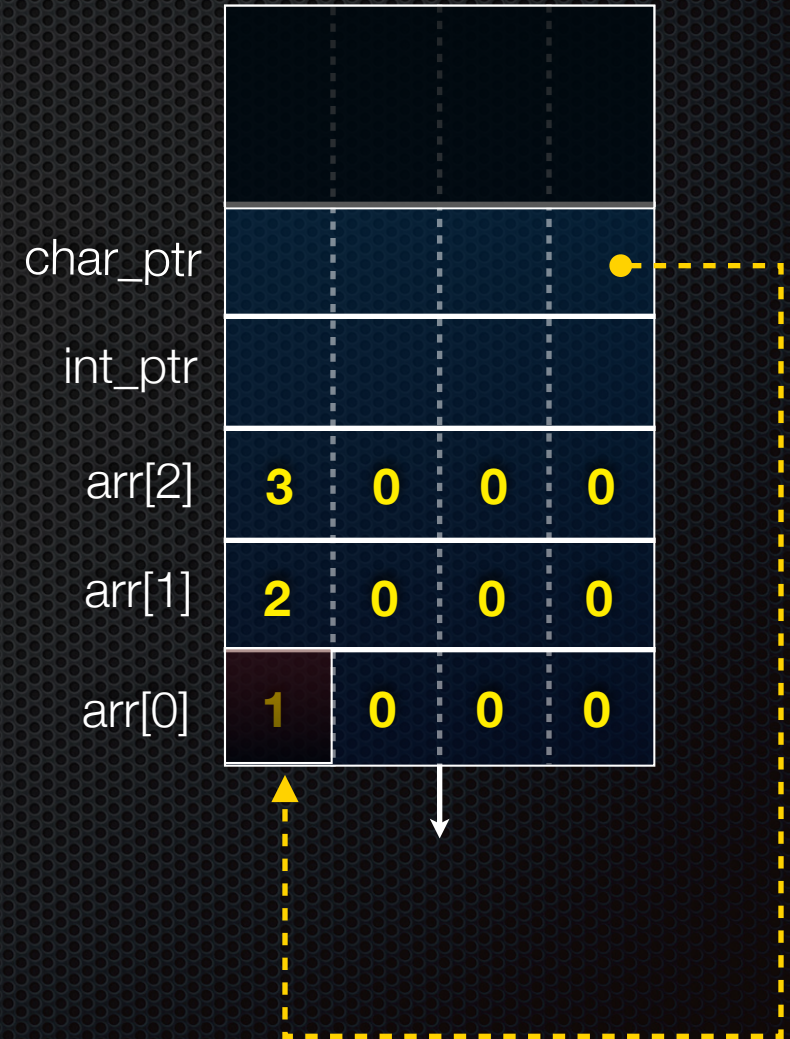
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

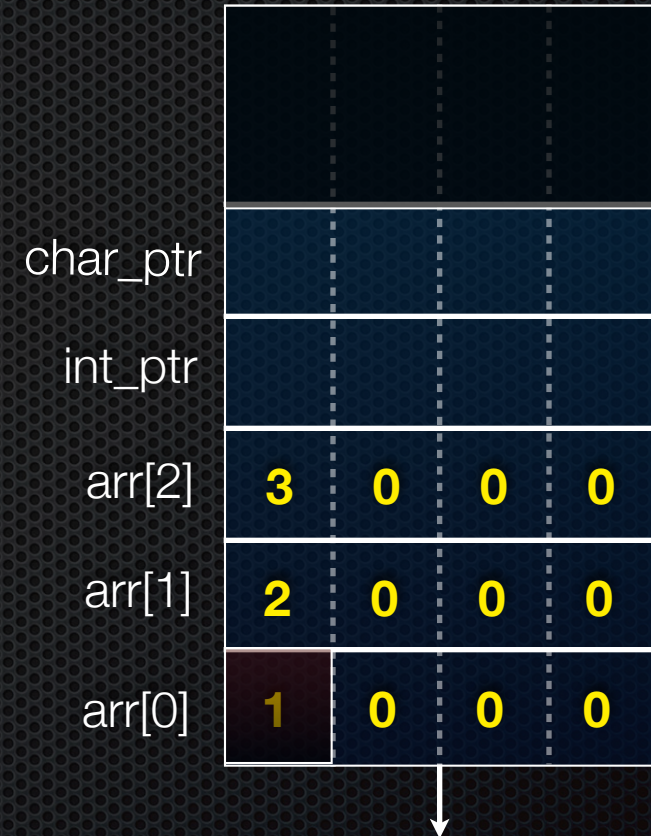
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



int\_ptr: 0xbffff2ac; \*int\_ptr: 1



```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



int\_ptr: 0xbffff2ac; \*int\_ptr: 1

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



int\_ptr: 0xbffff2ac; \*int\_ptr: 1

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

int_ptr: 0xbffff2ac; *int_ptr: 1
int_ptr: 0xbffff2b0; *int_ptr: 2

```

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

int_ptr: 0xbffff2ac; *int_ptr: 1
int_ptr: 0xbffff2b0; *int_ptr: 2

```

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

int_ptr: 0xbffff2ac; *int_ptr: 1
int_ptr: 0xbffff2b0; *int_ptr: 2
int_ptr: 0xbffff2b8; *int_ptr:
-1073745224

```

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

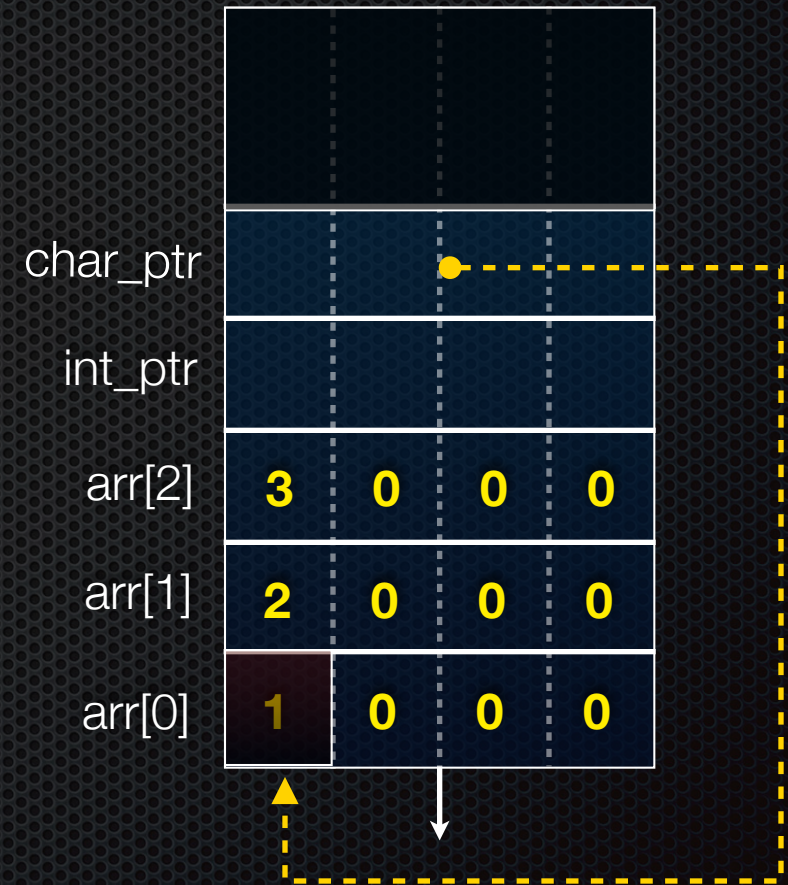
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

stack  
(assume 32-bit x86)



char\_ptr: 0xbffff2ac; \*char\_ptr: 1

pointerarithmetic.c

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

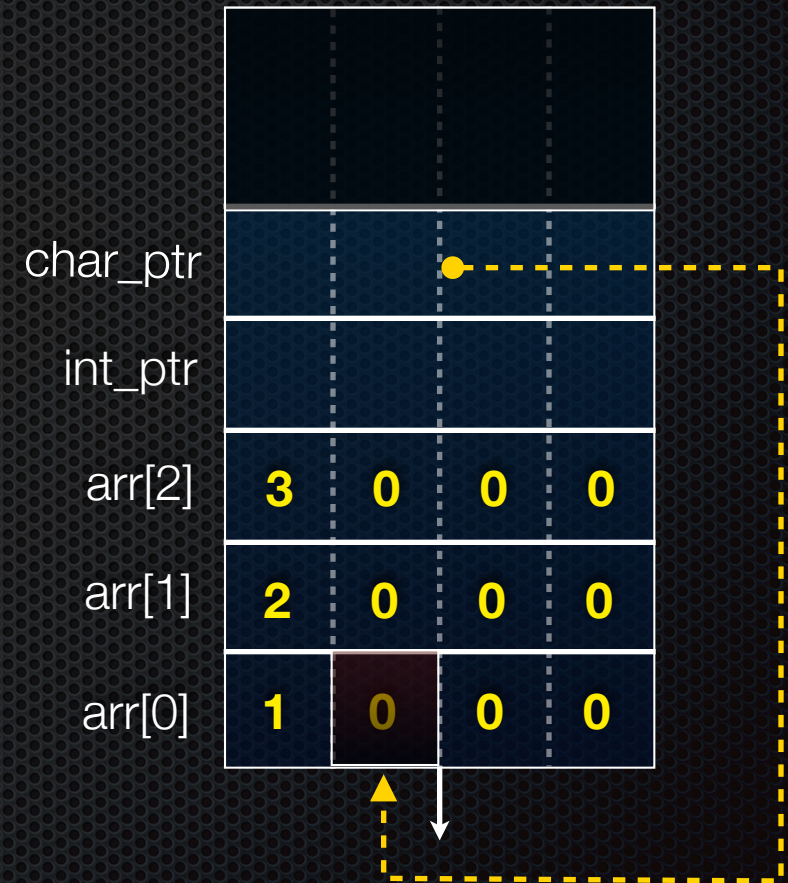
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



char\_ptr: 0xbffff2ac; \*char\_ptr: 1

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

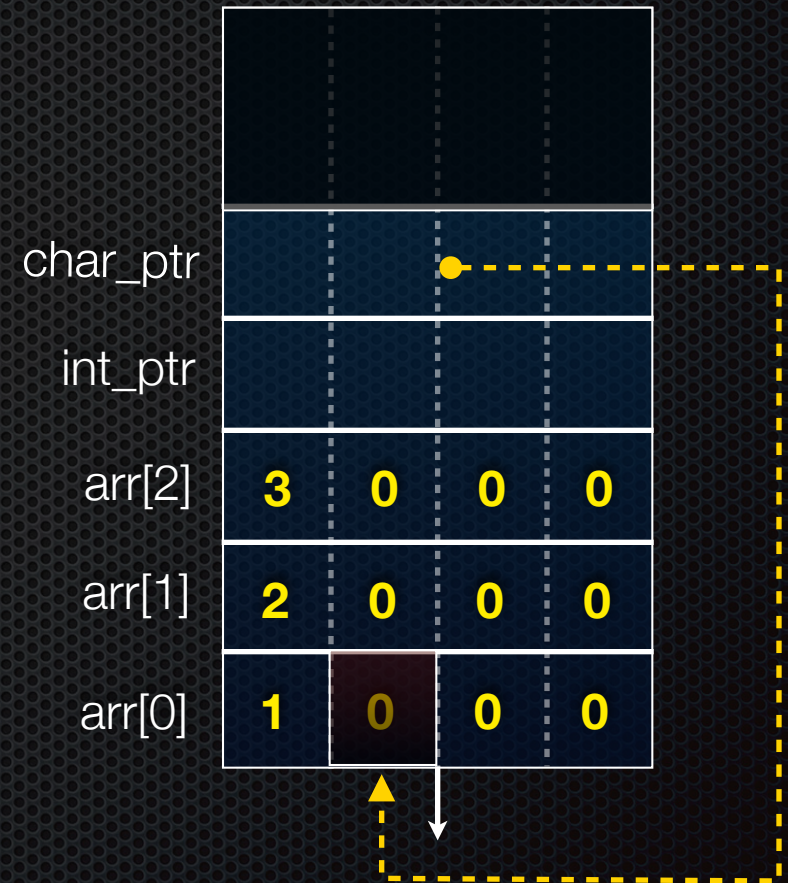
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

char_ptr: 0xbffff2ac; *char_ptr: 1
char_ptr: 0xbffff2ad; *char_ptr: 0

```



```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

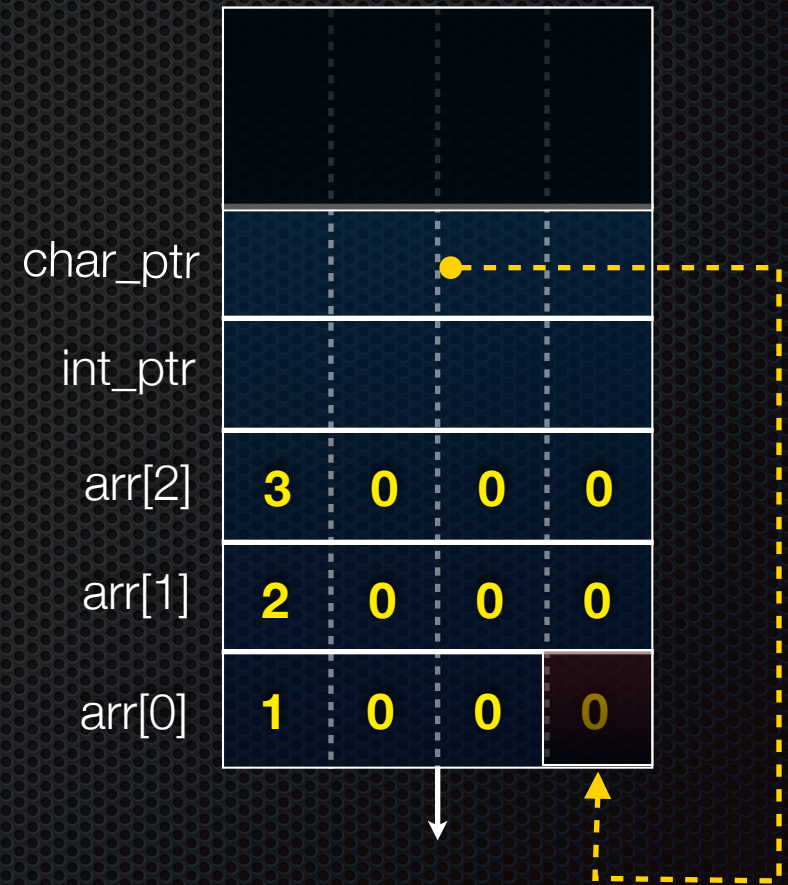
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

char_ptr: 0xbffff2ac; *char_ptr: 1
char_ptr: 0xbffff2ad; *char_ptr: 0

```

```

#include <stdio.h>

int main(int argc, char **argv) {
    int arr[3] = {1, 2, 3};
    int *int_ptr = &arr[0];
    char *char_ptr = (char *) int_ptr;

    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

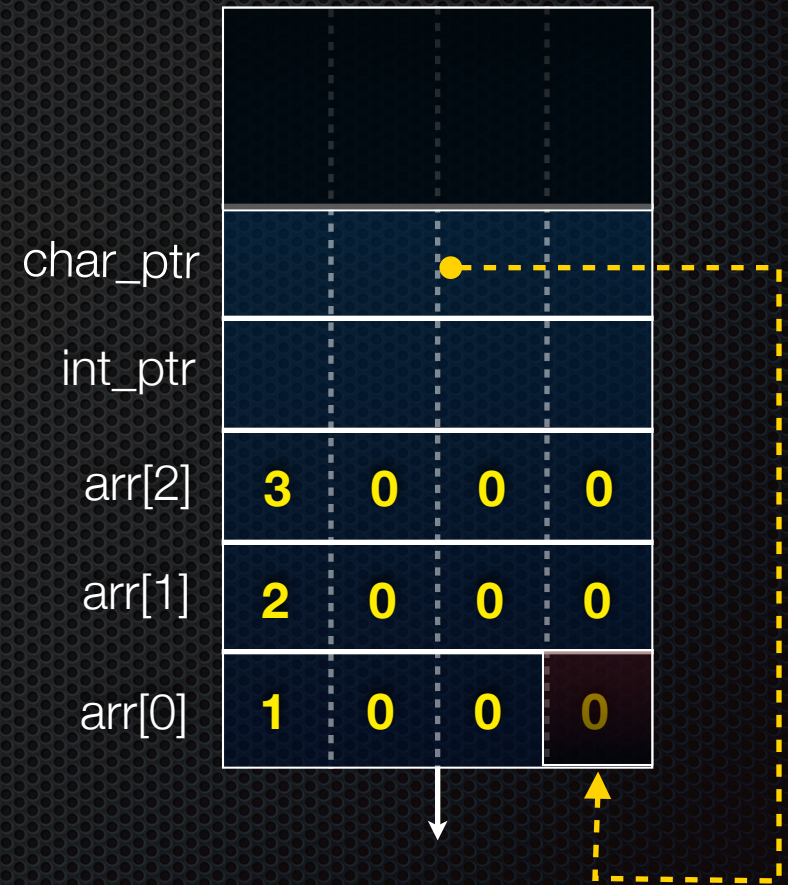
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack  
(assume 32-bit x86)



```

char_ptr: 0xbffff2ac; *char_ptr: 1
char_ptr: 0xbffff2ad; *char_ptr: 0
char_ptr: 0xbffff2af; *char_ptr: 0

```

# Pass-by-value

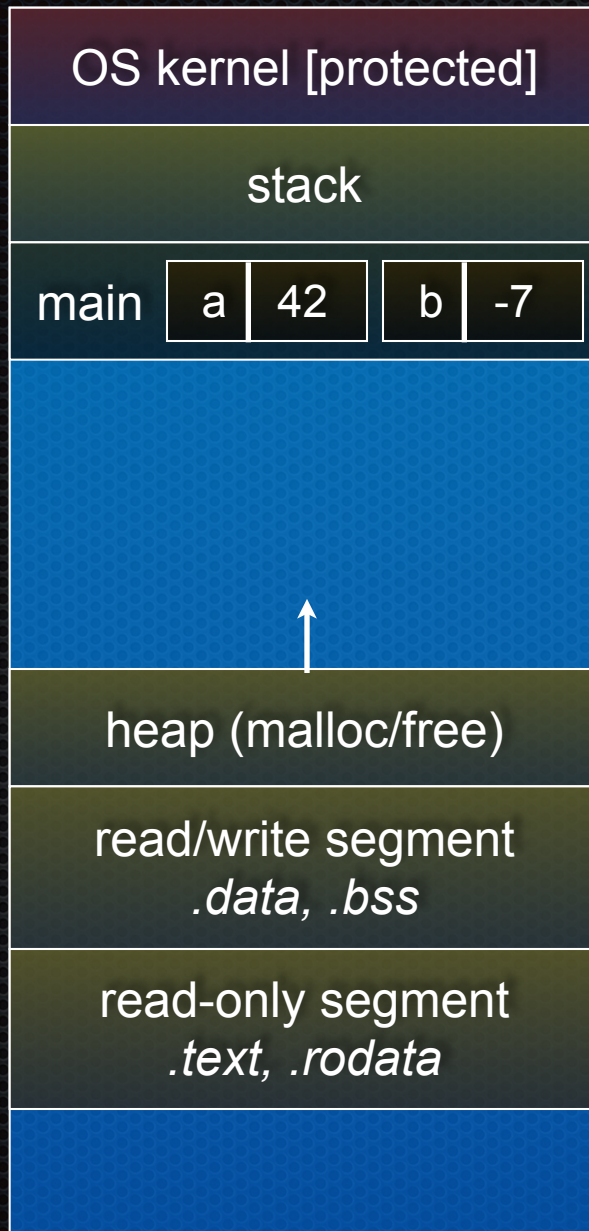
C passes arguments by **value**

- callee receives a **copy** of the argument
- if the callee modifies an argument, caller's copy isn't modified

```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

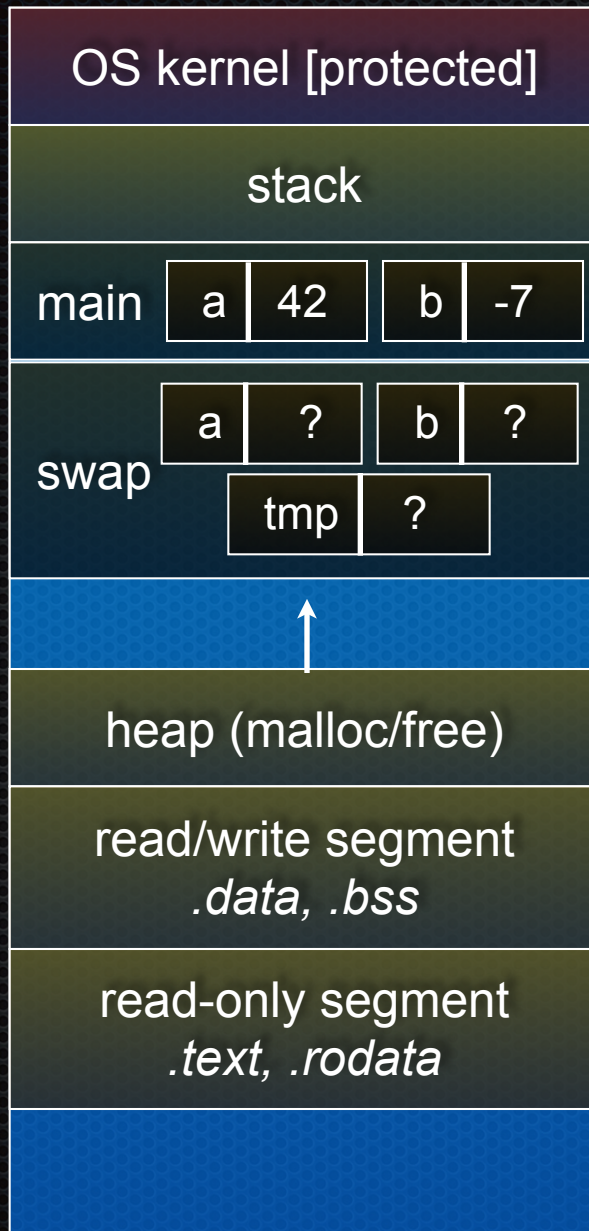
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

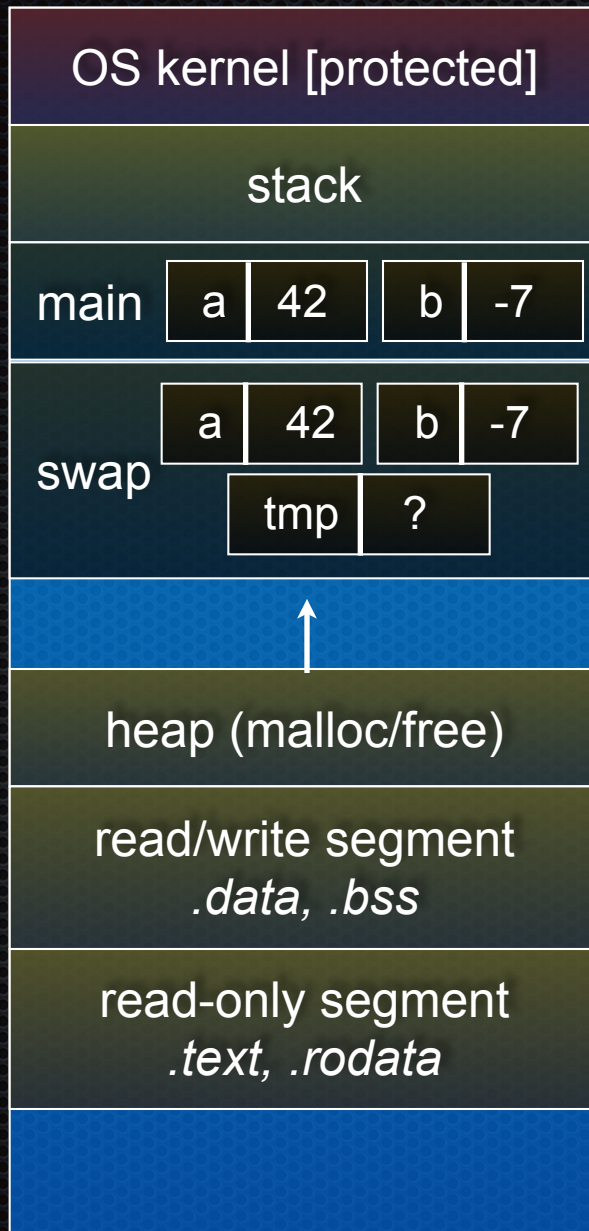
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

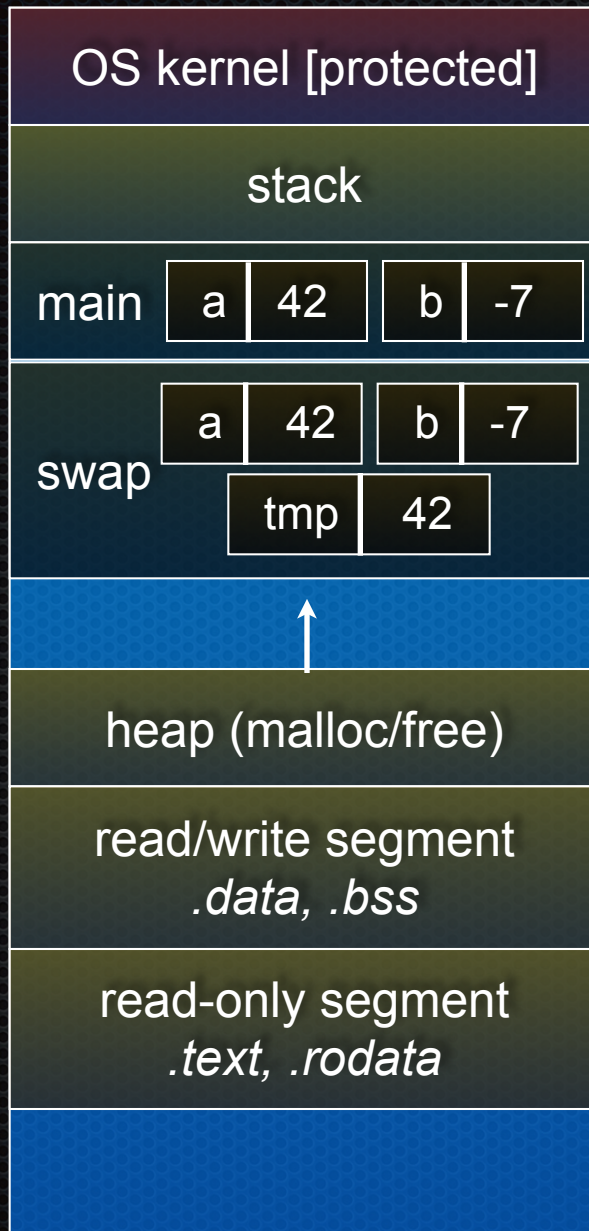
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

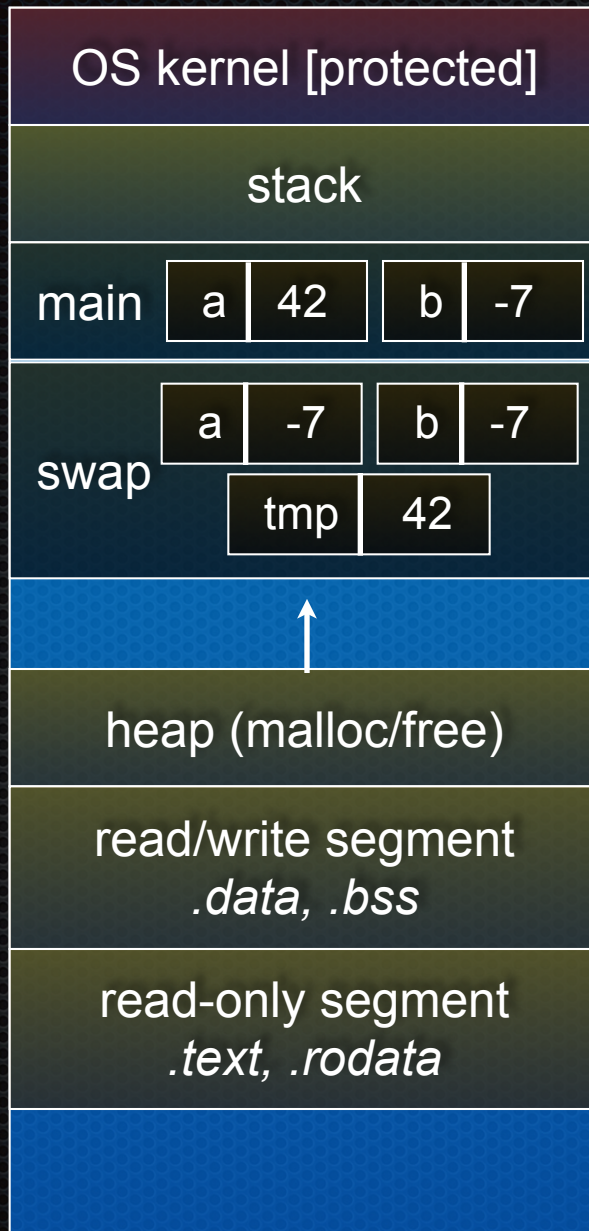
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

# Pass-by-value (stack)

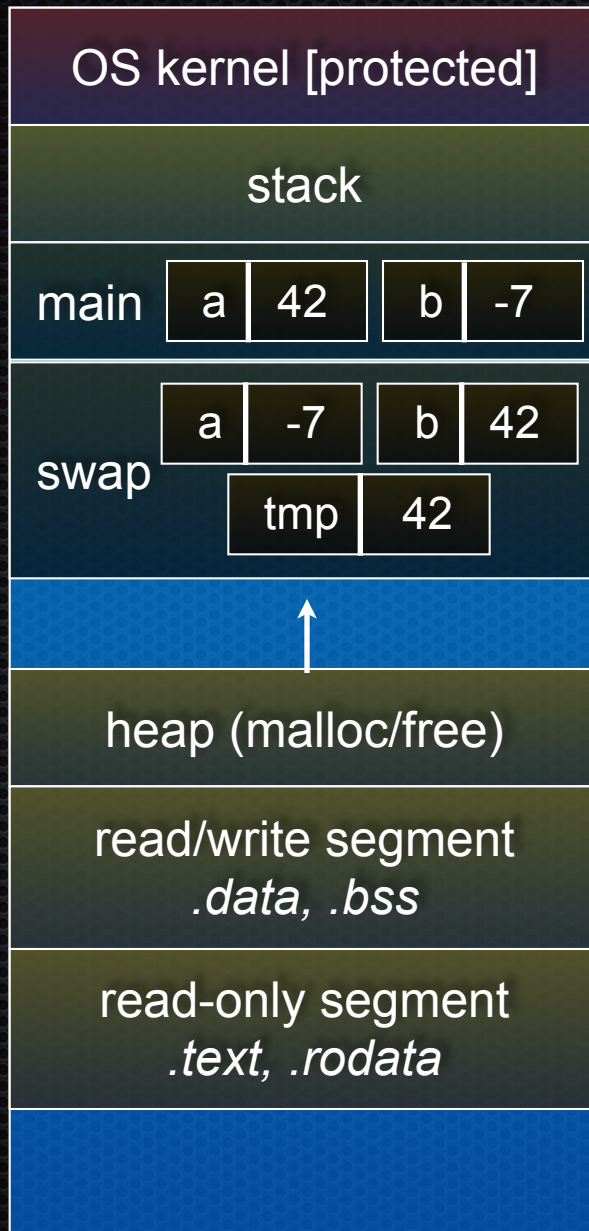


```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c



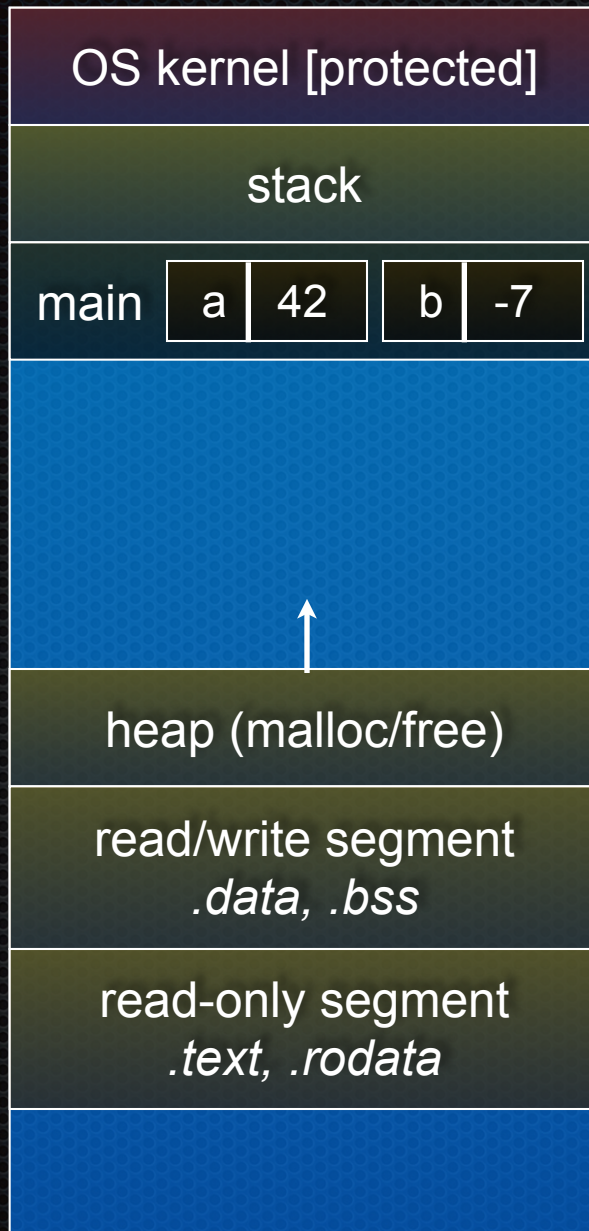
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

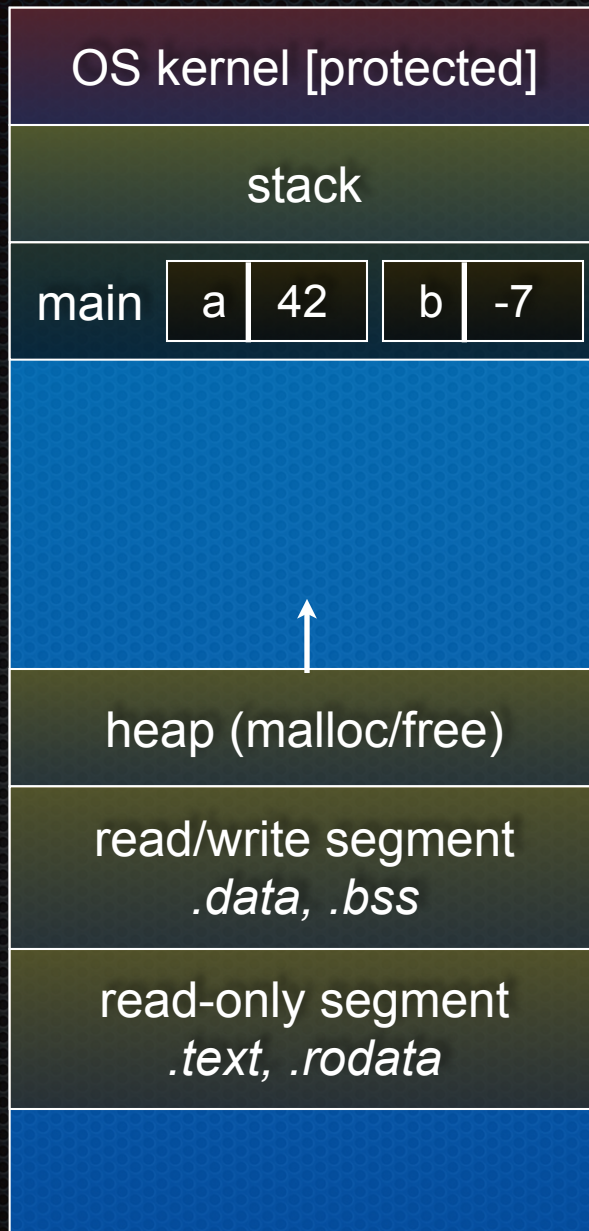
# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

# Pass-by-value (stack)



```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(a, b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

brokenswap.c

# Pass-by-reference

You can use pointers to pass by **reference**

- callee still receives a **copy** of the argument
  - ▶ but, the argument is a pointer
    - (so it's really call-by-value, but the value is a pointer)
  - ▶ the pointer's value points-to the variable in the scope of the caller
- this gives the callee a way to modify a variable that's in the scope of the caller

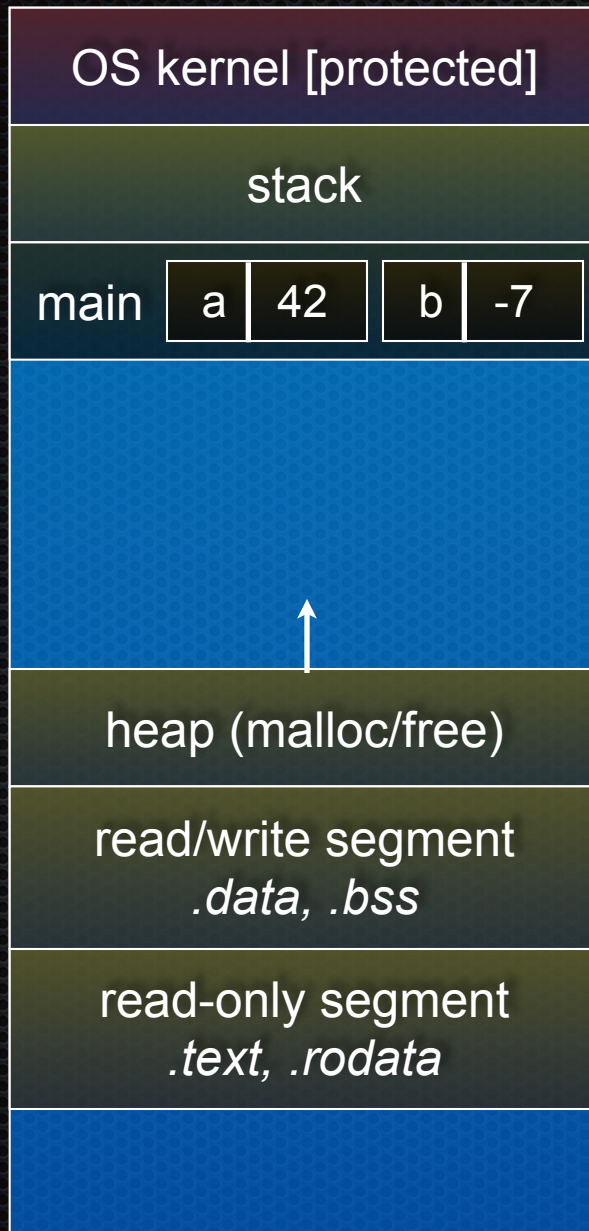
```
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(int argc, char **argv) {
    int a = 42, b = -7;

    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

swap.c

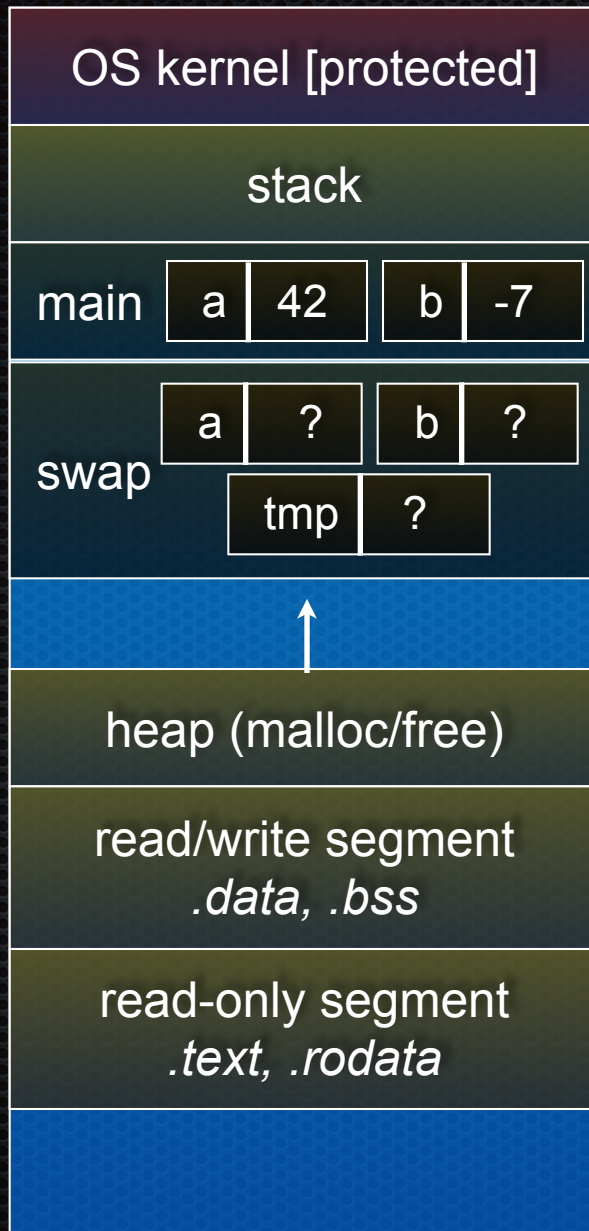
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

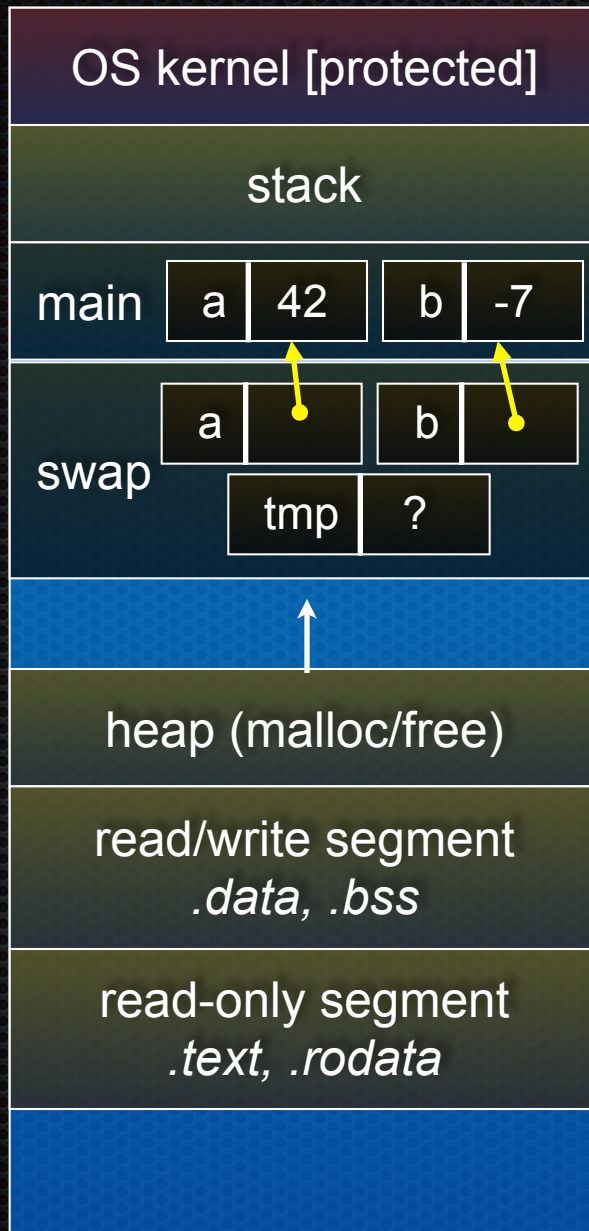
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

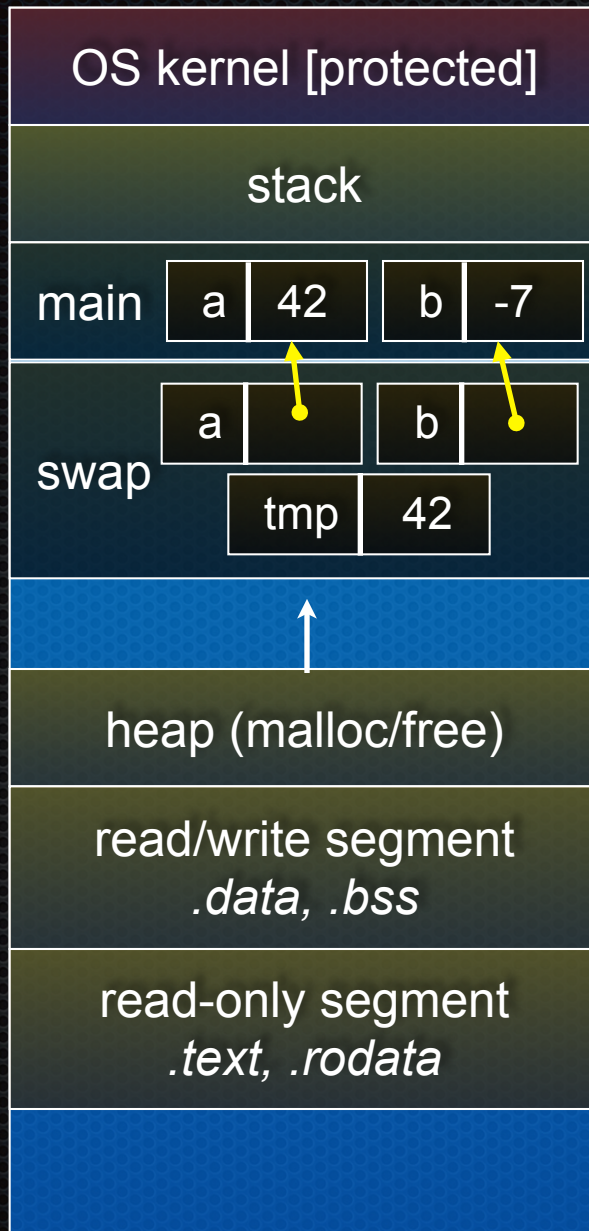
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

# Pass-by-reference (stack)

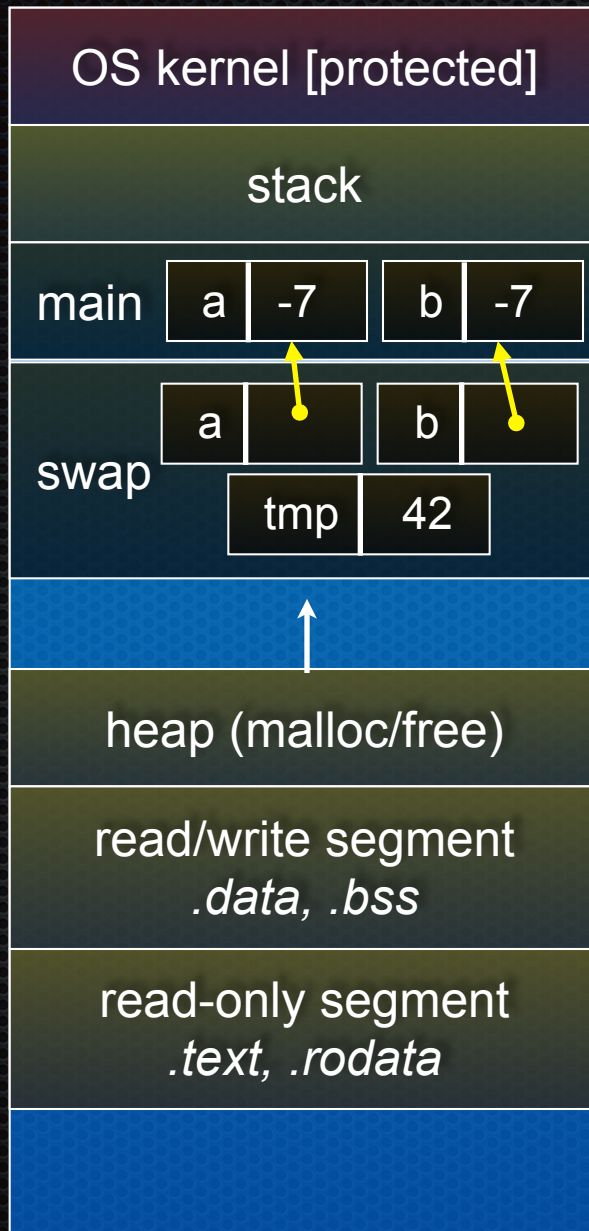


```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c



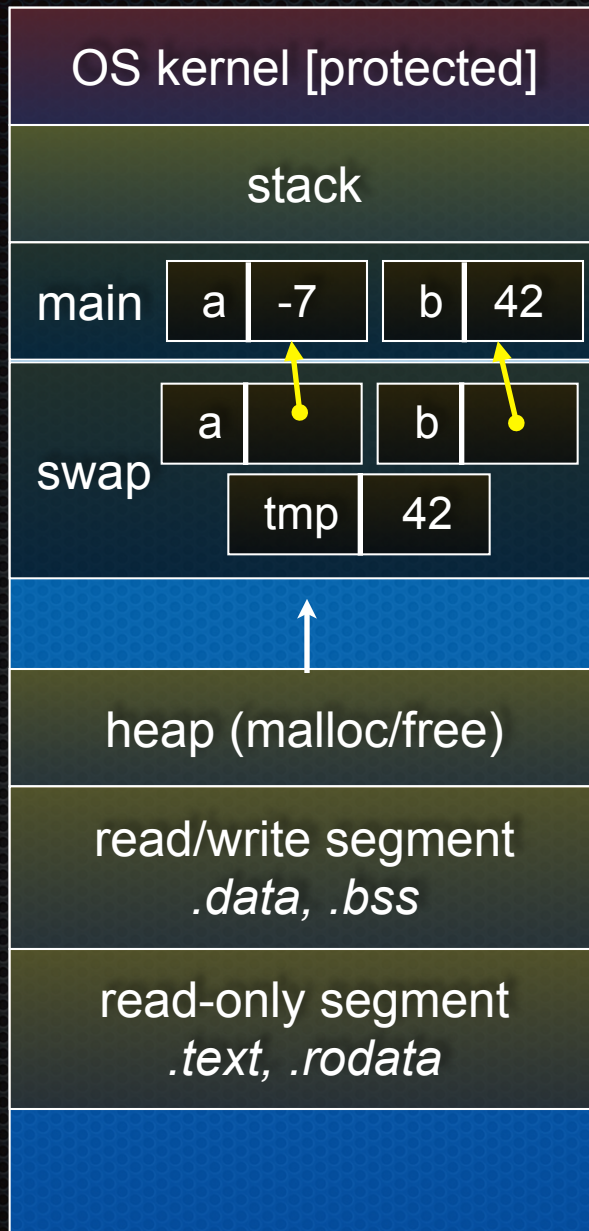
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

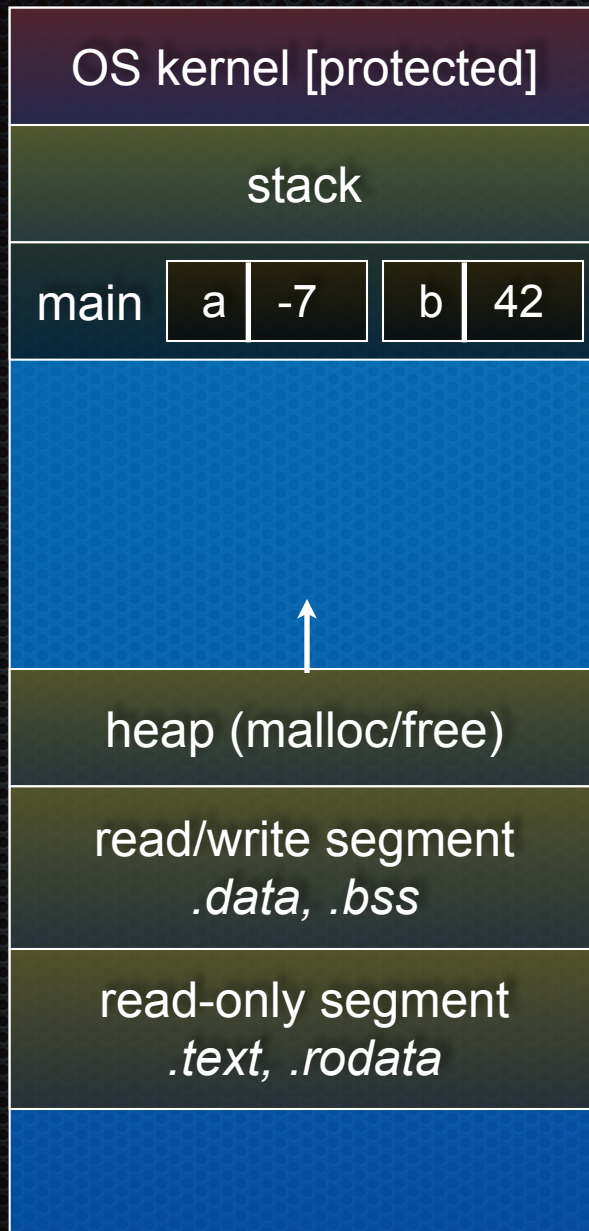
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

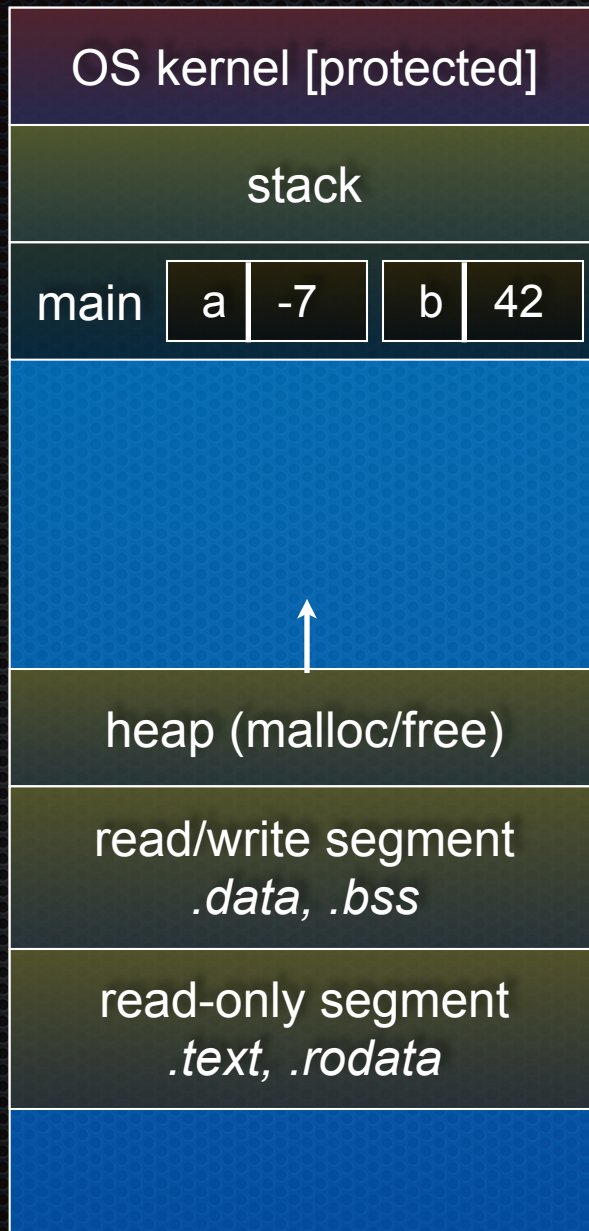
# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```

swap.c

# Pass-by-reference (stack)



```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}  
  
int main(int argc, char **argv) {  
    int a = 42, b = -7;  
  
    swap(&a, &b);  
    printf("a: %d, b: %d\n", a, b);  
    return 0;  
}
```



swap.c

# Arrays and pointers

a pointer can point to an array element

- an array's name can be used as a pointer to its first element
- and, you can use `[]` notation to treat a pointer like an array
  - ▶ `pointer[i]` is `i` elements' worth of bytes forward from pointer

```
int a[5] = {10, 20, 30, 40, 50};
int* p1 = &a[3];    // refers to a's fourth element
int* p2 = &a[0];    // refers to a's first element
int* p3 = a;        // refers to a's first element

*p1 = 100;
*p2 = 200;
p1[1] = 300;
p2[1] = 400;
p3[2] = 500;        // final: 200, 400, 500, 100, 300
```

# Passing arrays as parameters

array parameters are really passed as pointers to the first array element

- the `[]` syntax for parameter types is just for convenience

```
void f(int a[]);

int main(...) {
    int a[5];
    ...
    f(a);
    return 0;
}

void f(int a[] ){
```

your code

```
void f(int *a);

int main(...) {
    int a[5];
    ...
    f(&a[0]);
    return 0;
}

void f(int *a) {
```

equivalent to

# Self-Exercise 1

Use a box-and-arrow diagram for the following program to explain what it prints out:

```
#include <stdio.h>

int foo(int *bar, int **baz) {
    *bar = 5;
    *(bar+1) = 6;
    *baz = bar+2;
    return *((*baz)+1);
}

int main(int argc, char **argv) {
    int arr[4] = {1, 2, 3, 4};
    int *ptr;

    arr[0] = foo(&(arr[0]), &ptr);
    printf("%d %d %d %d %d\n",
           arr[0], arr[1], arr[2], arr[3], *ptr);
    return 0;
}
```

# Self-Exercise 2

Write a program that prints out whether the computer it is running on is little endian or big endian.

- (hint: see pointerarithmetic.c from today's lecture)



# Self-Exercise 3

Write a function that:

- accepts an (array of ints) and an (array length) as arguments
- malloc's an (array of (int \*)) of the same length
- initializes each element of the newly allocated array to point to the corresponding element in the passed-in array
- returns a pointer to the newly allocated array

See you on Monday!