

CSE 333

Lecture 4 - pointers, pointers, pointers

Steve Gribble

Department of Computer Science & Engineering

University of Washington



Administrivia

Take a look at the course schedule and note that:

- self-exercise solutions are up
- assignment #1 is up
 - ▶ due in 10 days: Thursday April 12th by **6:00pm**
 - ▶ work individually, not in teams of 2
- assignment #1.5 (another survey) is up
- no lecture on Wednesday April 25th
 - ▶ Steve is traveling

Exercises

Sorry for all the confusion! A few things to note:

- you should try compiling on attu or a home VM
 - ▶ Linux's compiler issues more warnings than Mac OS X's
 - ▶ we will treat attu / home VM as the bar for you to clear
- we will be compiling with the following
 - ▶ `gcc -Wall -std=c99 -o ex3 ex3.c`
- going forward, no more amnesty for printf format warnings :)

Ex2 statistics

New grading scheme is helping

- 15 0's
- 10 1's
- 49 2's
- 4 3's

Still some people getting 0's because of warnings, code that doesn't compile, etc.

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; // x now contains the value 42
int *p; // p is a pointer to an integer
p = &x; // p now contains the address of x

printf("x is %d\n", x);

*p = 99; // store 99 where p points to
printf("x is %d\n");
```

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;

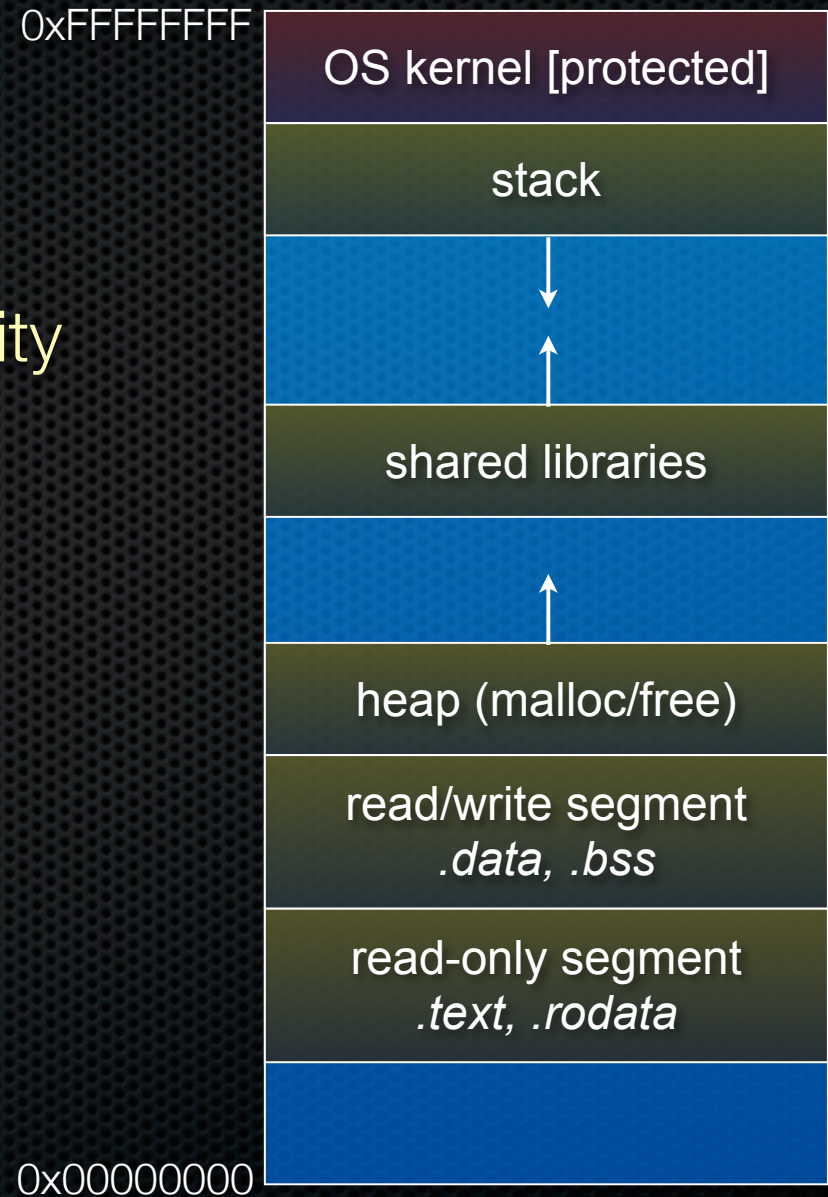
    printf("&x: %p; p: %p; &p: %p\n",
           &x, p, &p);
    return 0;
}
```

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

x	1
----------	---

0xbfff2d0

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

0xbfff2d4

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

0xbfff2d8

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

0xbfff2cc

p	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

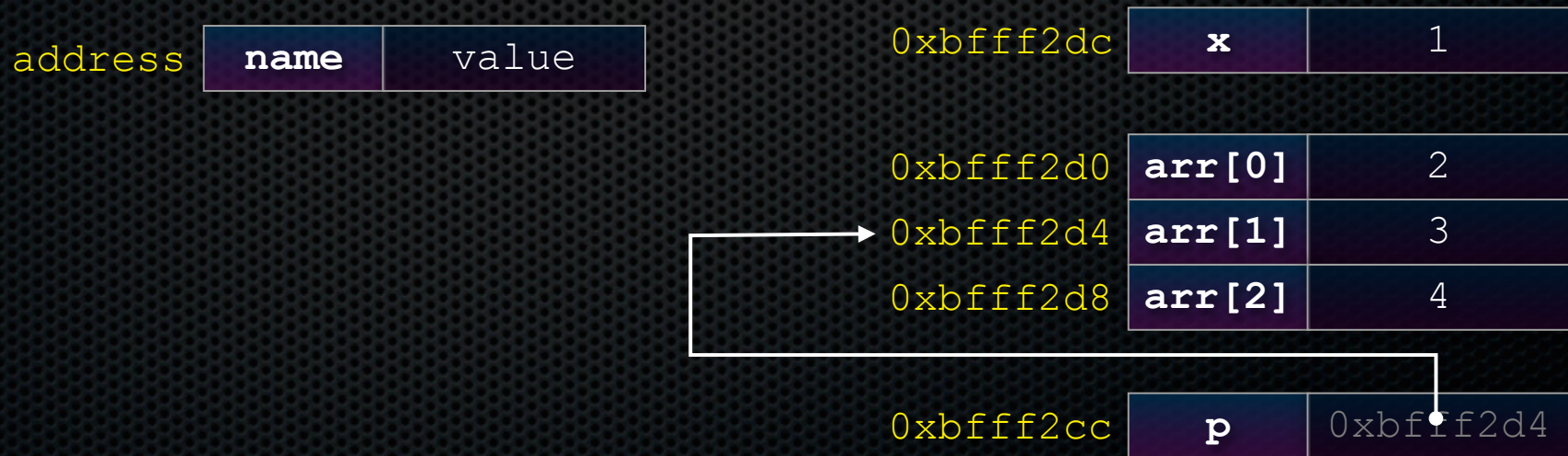
0xbfff2d4

main()'s stack frame

Box and arrow diagrams

boxarrow.c

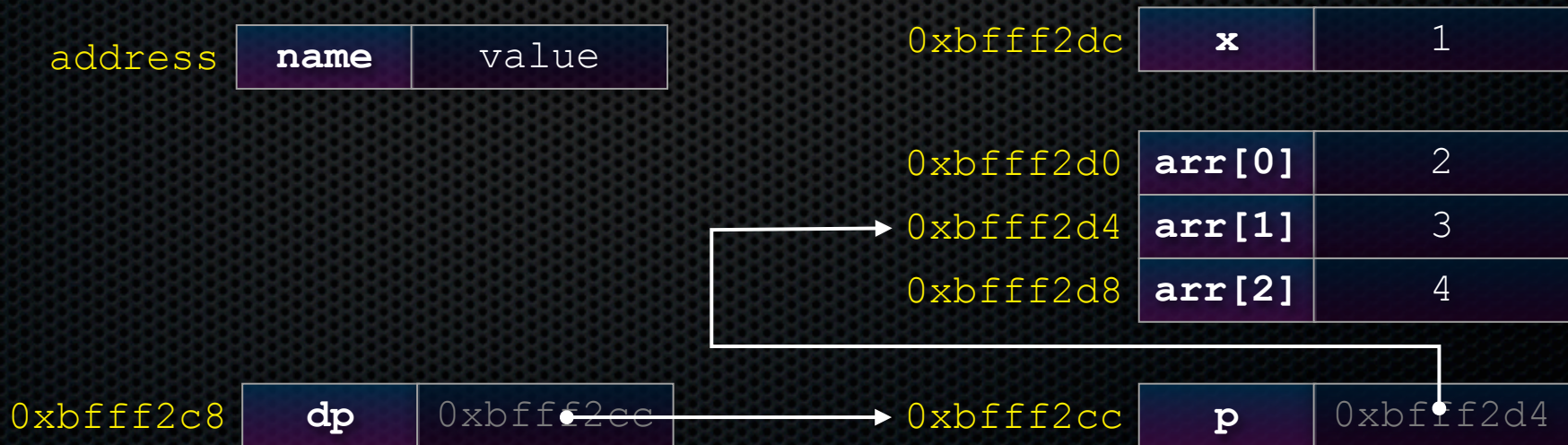
```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
  
    printf("&x: %p; x: %d\n", &x, x);  
    printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);  
    printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);  
    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);  
    return 0;  
}
```



Box and arrow diagrams

boxarrow2.c

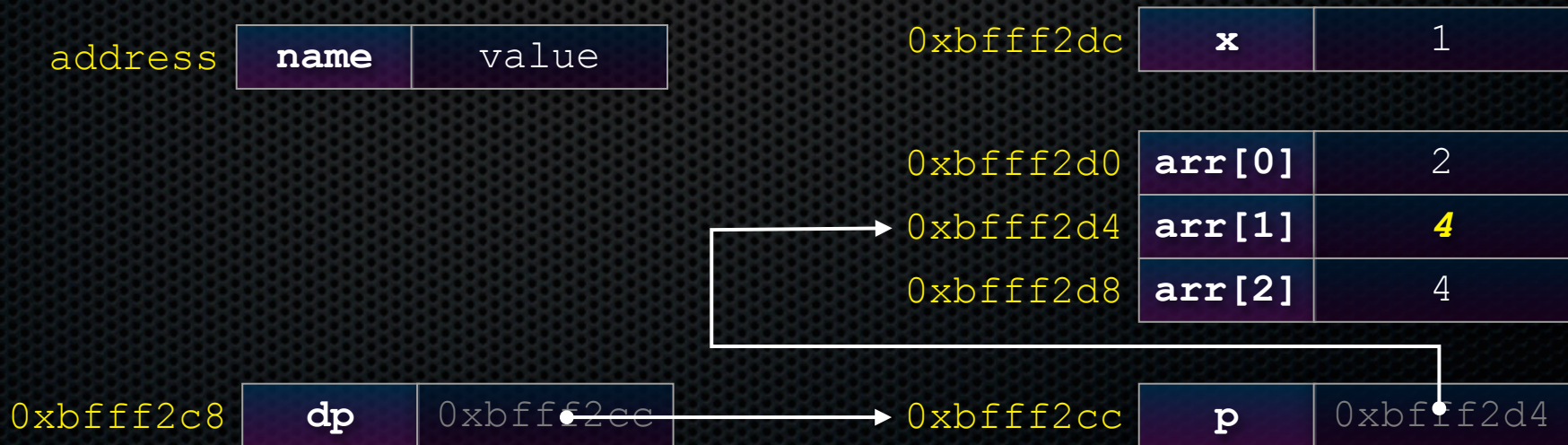
```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
   >(*dp) += 1;  
    p += 1;  
   >(*dp) += 1;  
    return 0;  
}
```



Box and arrow diagrams

boxarrow2.c

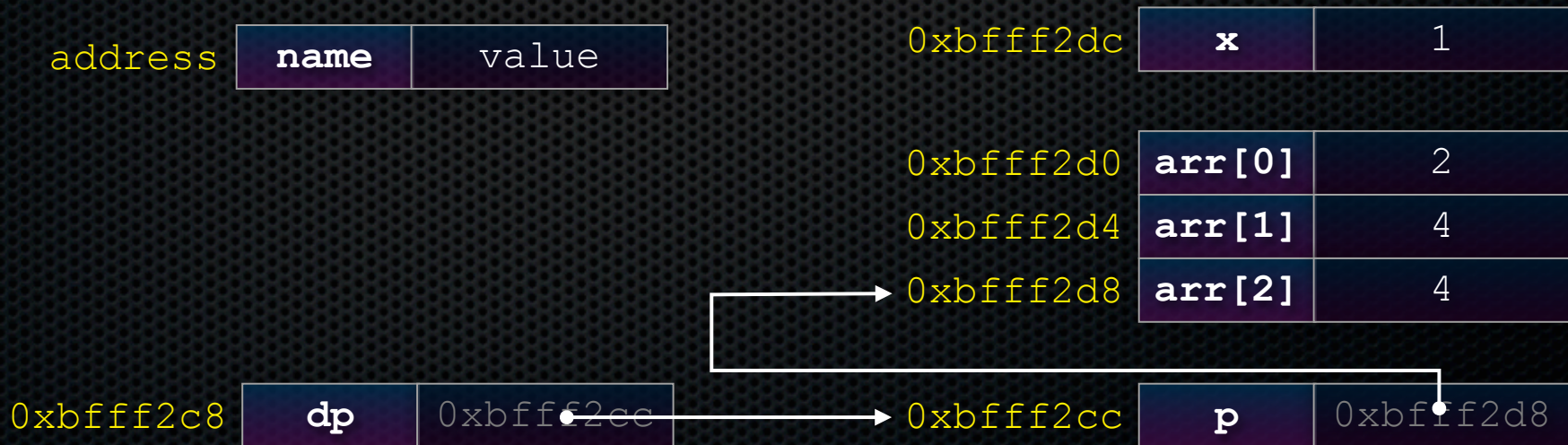
```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
    __(*dp) += 1;  
    p += 1;    // *dp += 1;  
    __(*dp) += 1;  
    return 0;  
}
```



Box and arrow diagrams

boxarrow2.c

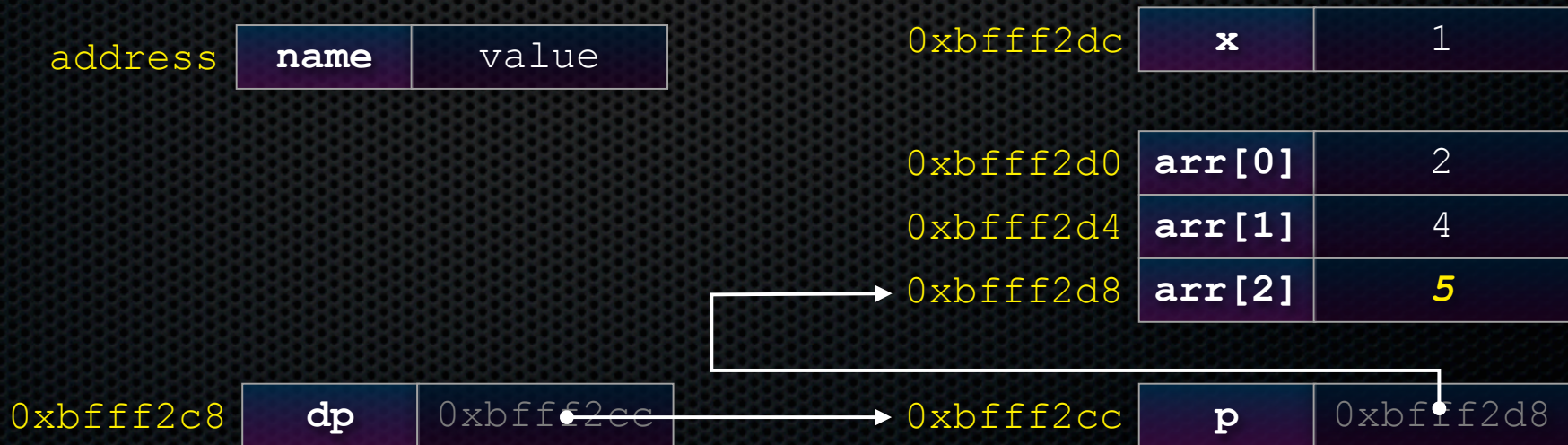
```
int main(int argc, char **argv) {  
    int x = 1;  
    int arr[3] = {2, 3, 4};  
    int *p = &arr[1];  
    int **dp = &p;  
  
   >(*dp) += 1;  
    p += 1;  
   >(*dp) += 1;  
    return 0;  
}
```



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;  
}
```



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



```

#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;

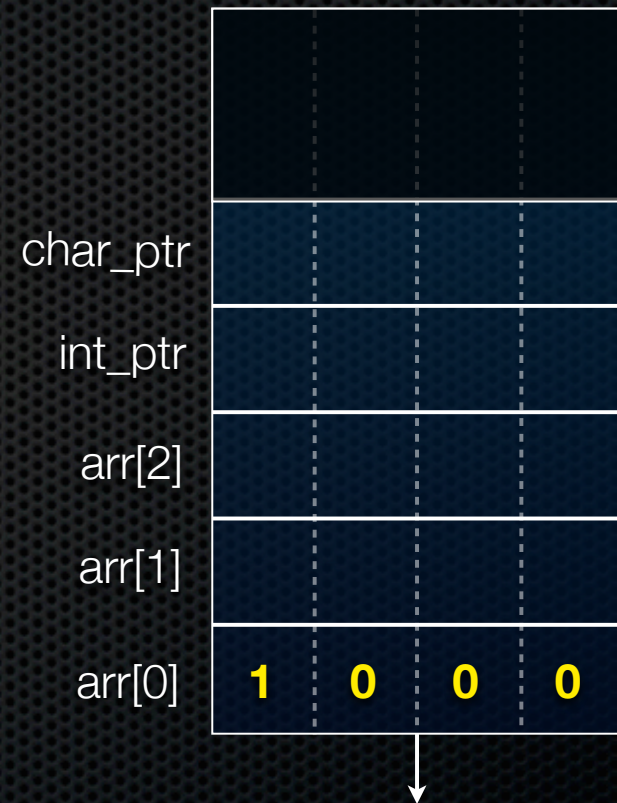
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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)



(x86 is little endian)

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;

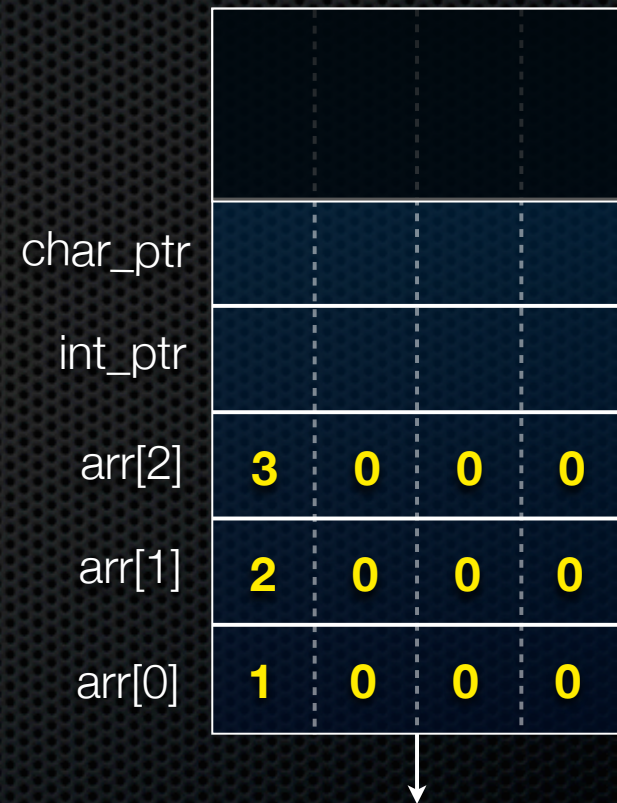
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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;
}

```

stack
(assume 32-bit x86)



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
    print("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)



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
    print("int_ptr: %p; *int_ptr: %d\n",
          int_ptr, *int_ptr);

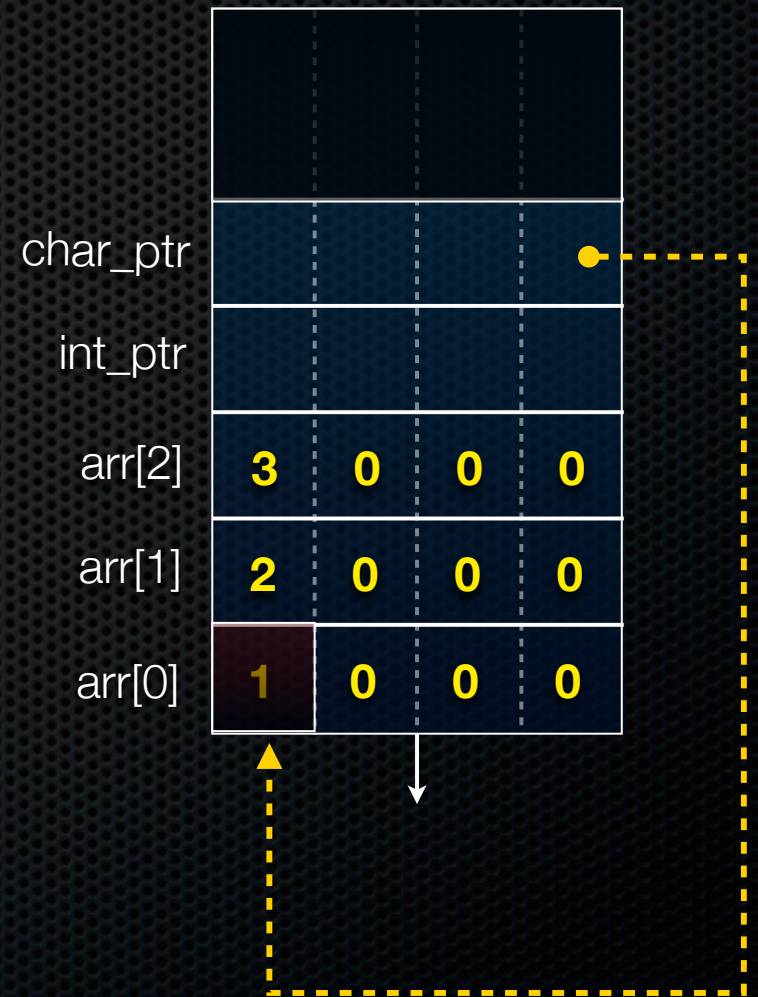
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    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;
}

```

stack
(assume 32-bit x86)



```

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

```

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;
}

```

stack
(assume 32-bit x86)



```

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

```

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)



```

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;

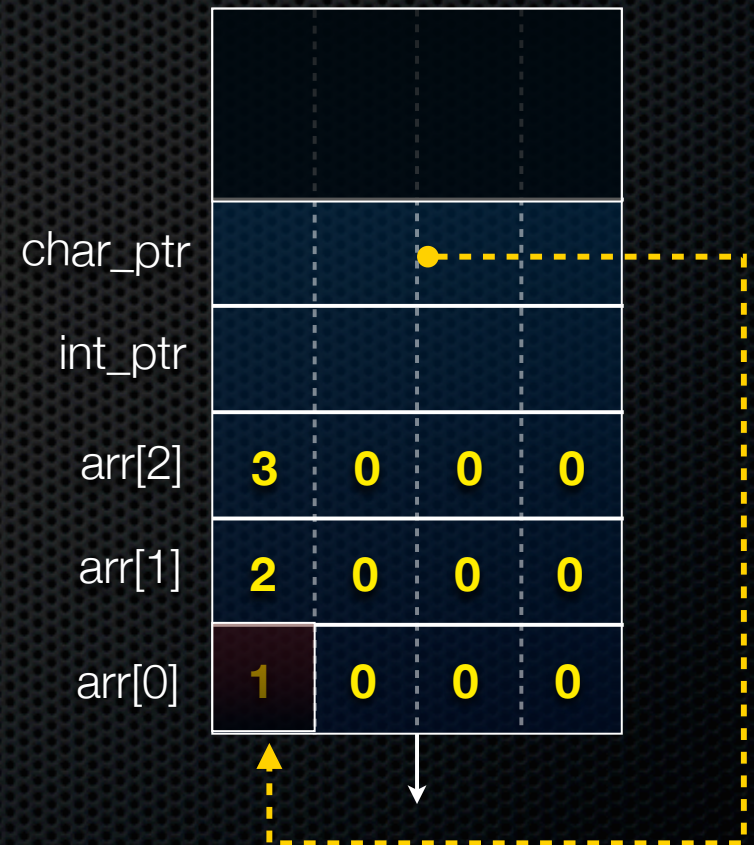
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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;

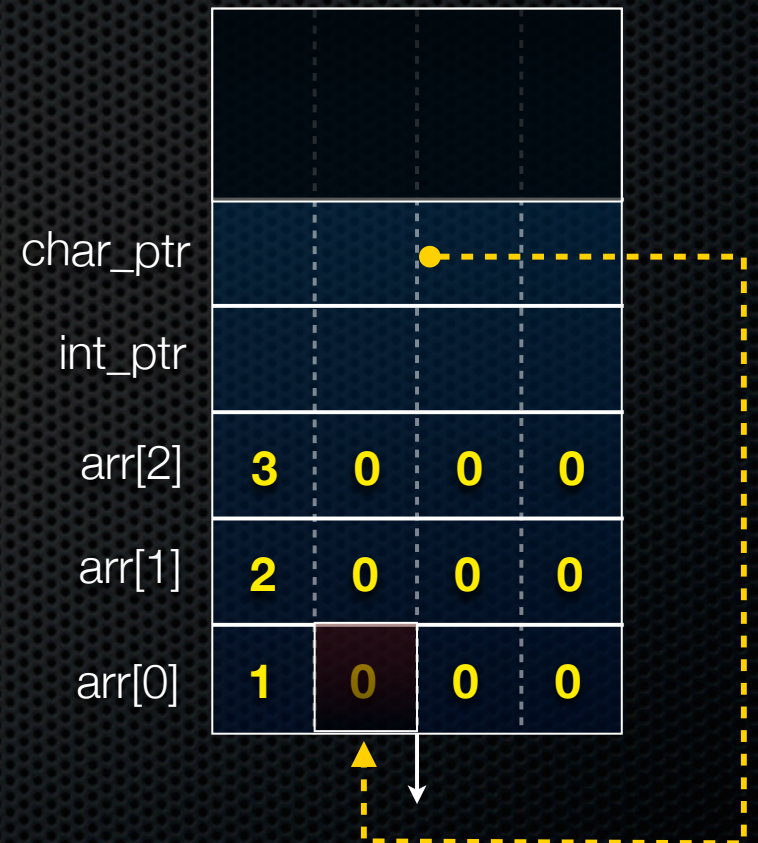
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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;
}

```

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;

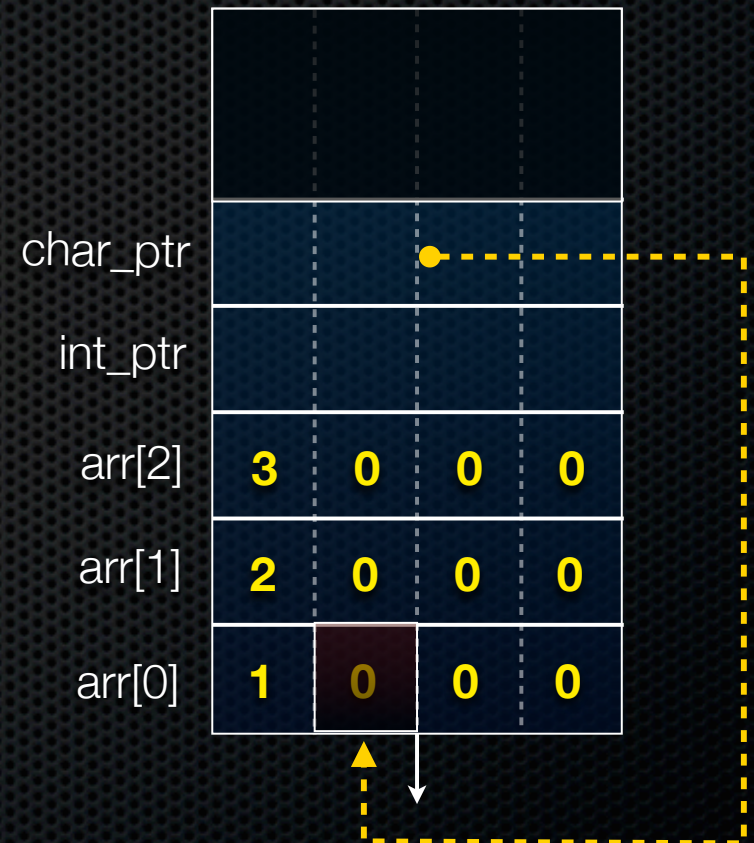
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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;
}

```

stack
(assume 32-bit x86)



```

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

```

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);

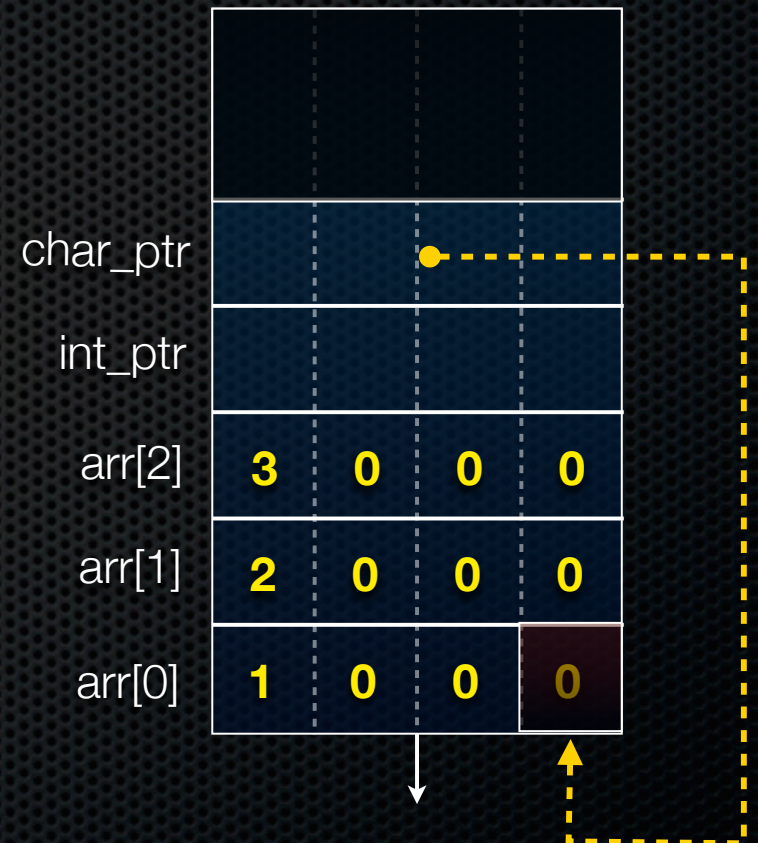
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 1;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);
    char_ptr += 2;
    printf("char_ptr: %p; *char_ptr: %d\n",
           char_ptr, *char_ptr);

    return 0;
}

```

pointerarithmetic.c

stack
(assume 32-bit x86)



```

char_ptr: 0xbffff2ac; *char_ptr: 1
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;

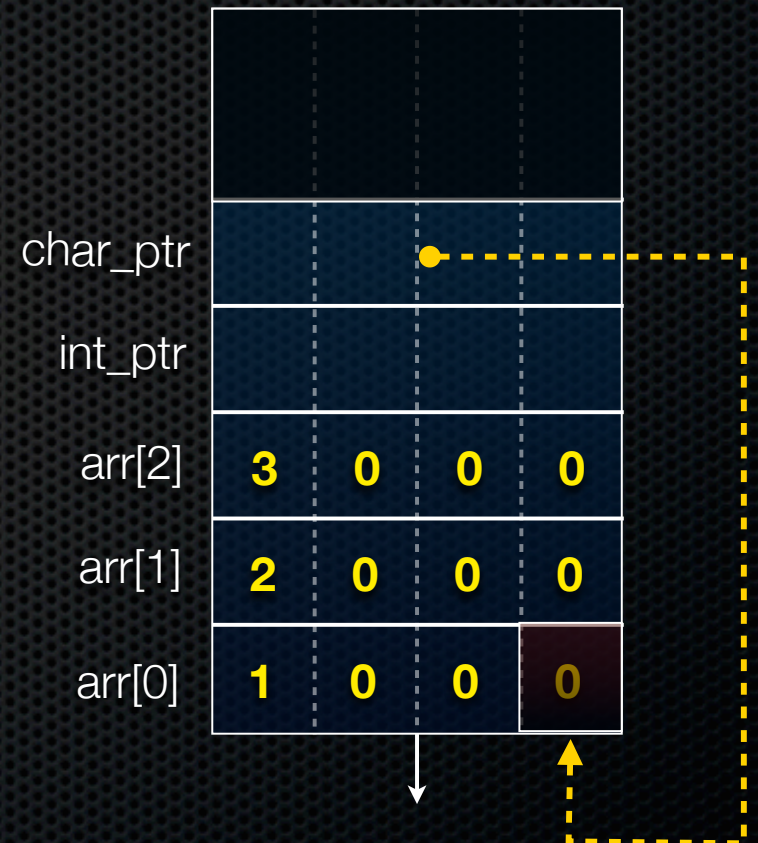
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 1;
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);
    int_ptr += 2; // uh oh
    printf("int_ptr: %p; *int_ptr: %d\n",
           int_ptr, *int_ptr);

    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;
}

```

stack
(assume 32-bit x86)



```

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

```

pointerarithmetic.c

Pass-by-value

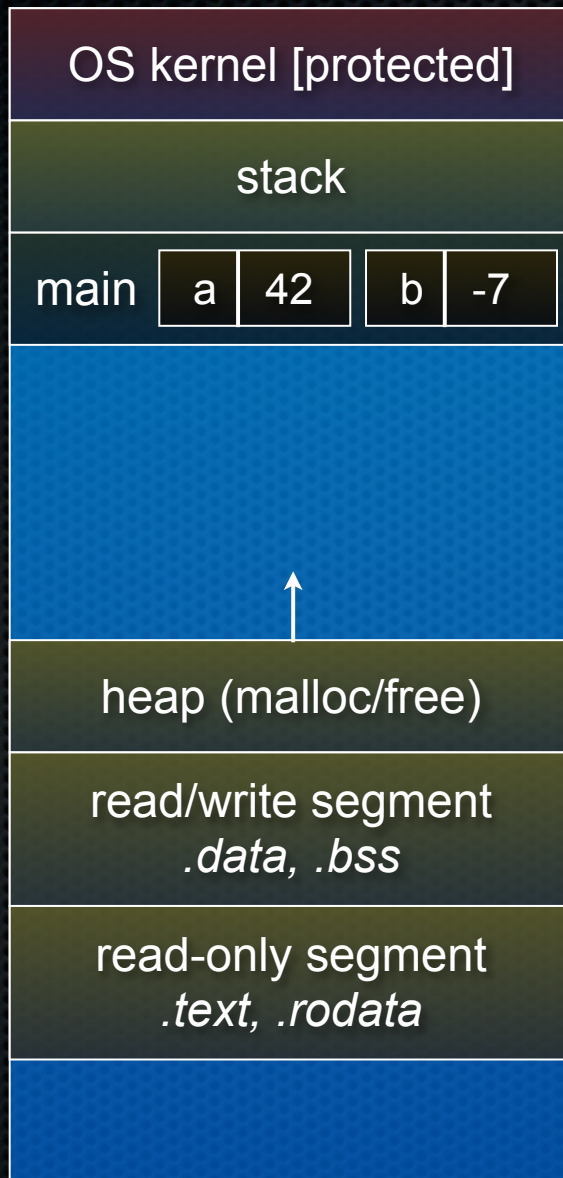
C passes arguments by **value**

- 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

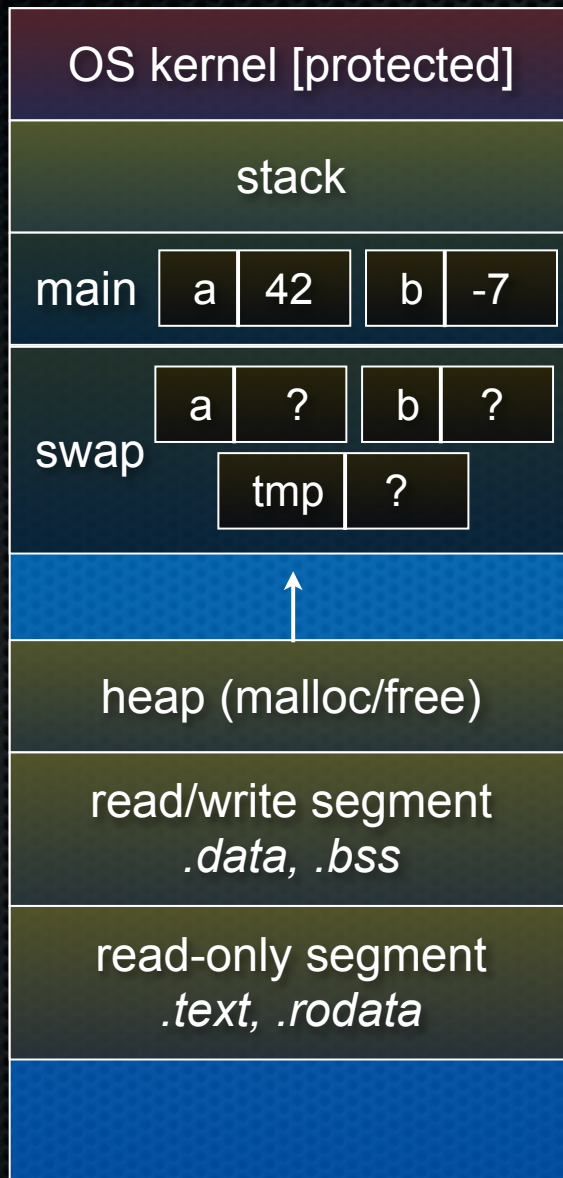
Pass-by-value (stack)



```
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

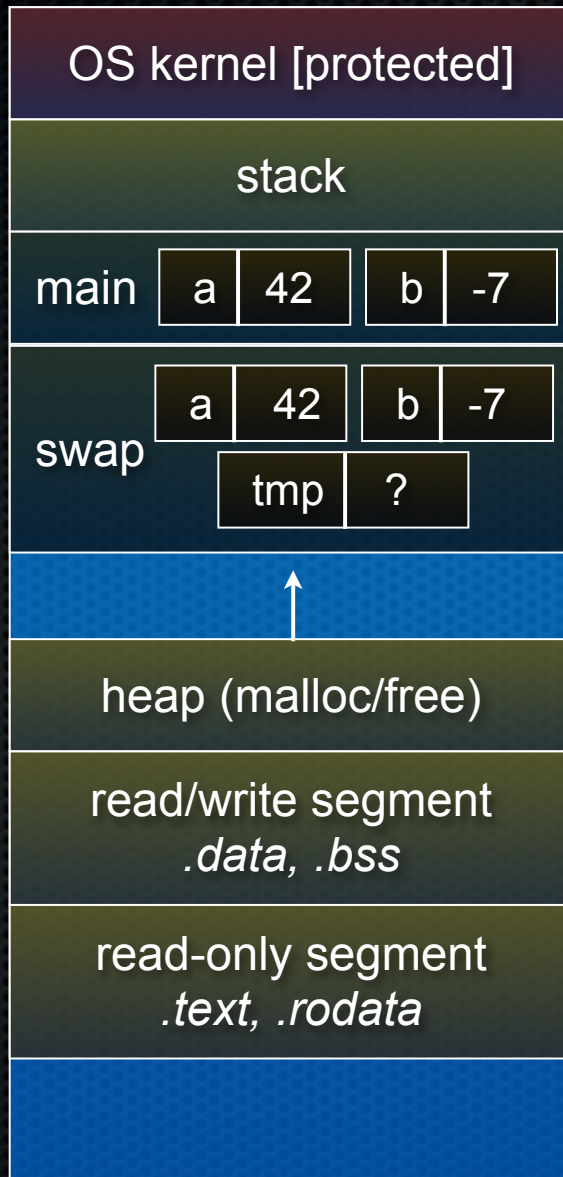
Pass-by-value (stack)



```
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

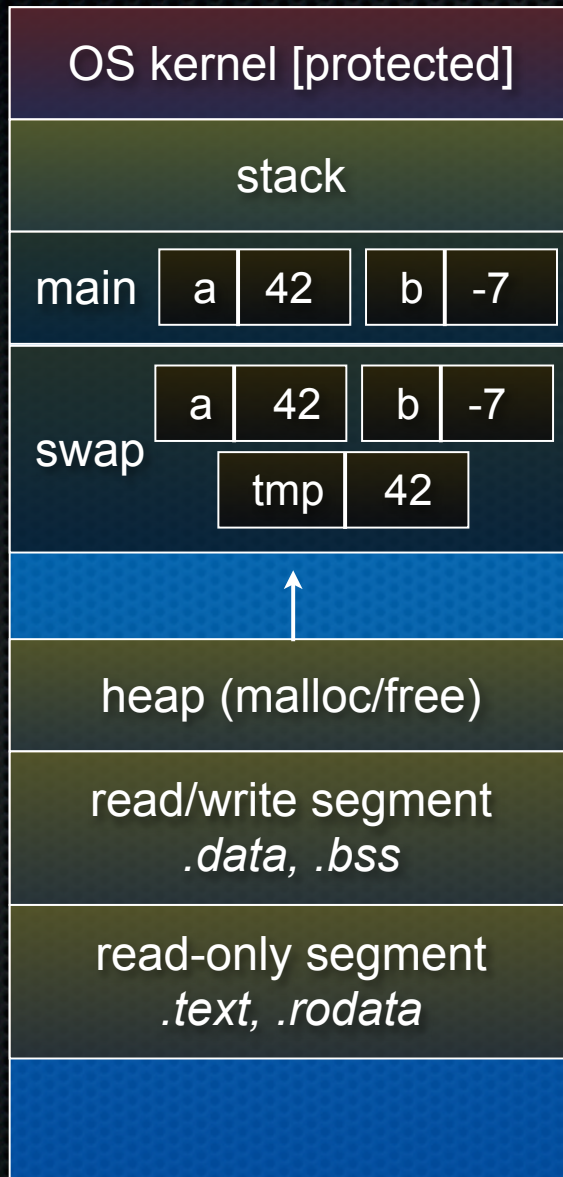
Pass-by-value (stack)



```
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

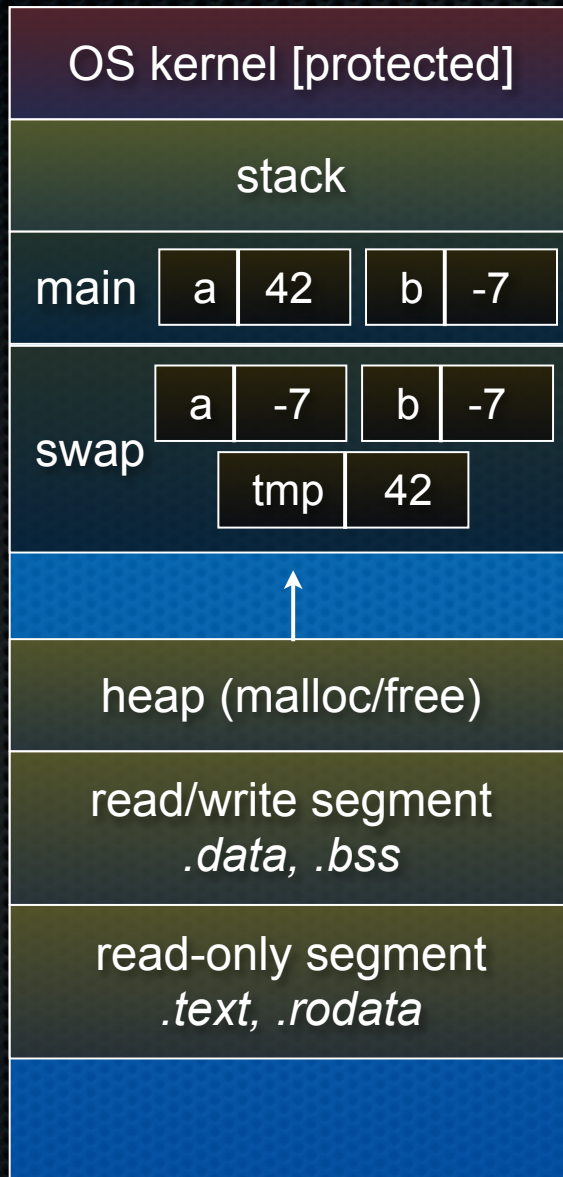
Pass-by-value (stack)



```
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

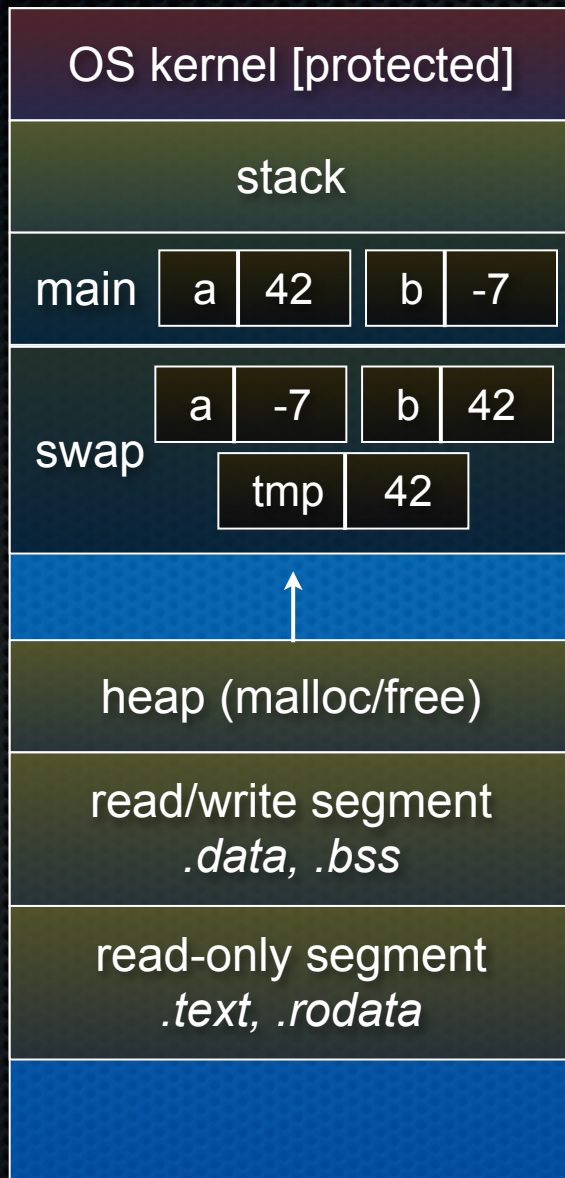
Pass-by-value (stack)



```
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

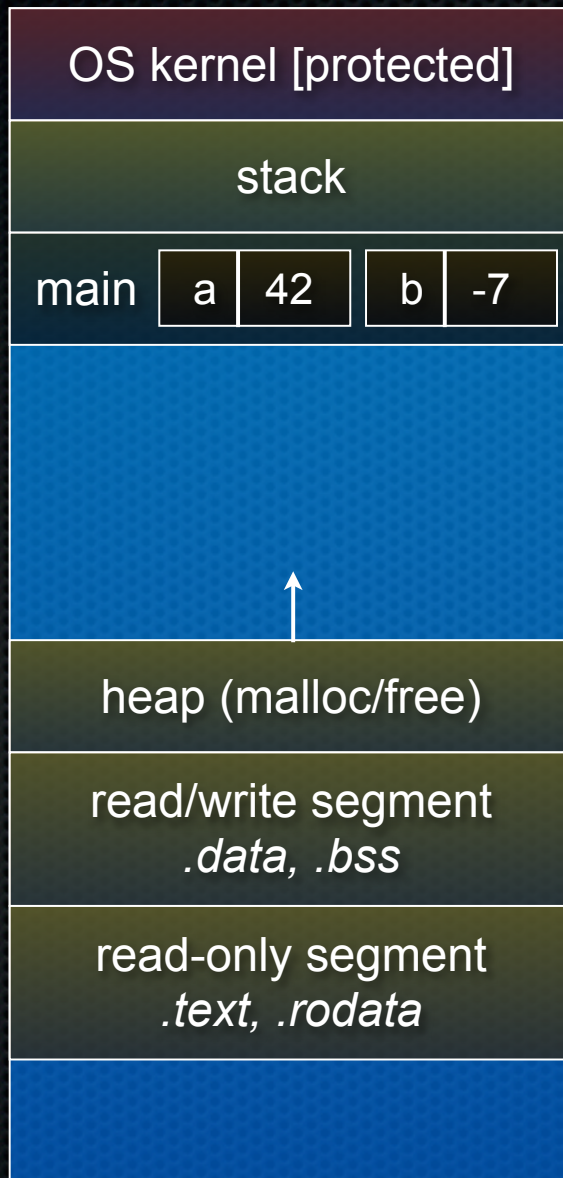
Pass-by-value (stack)



```
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

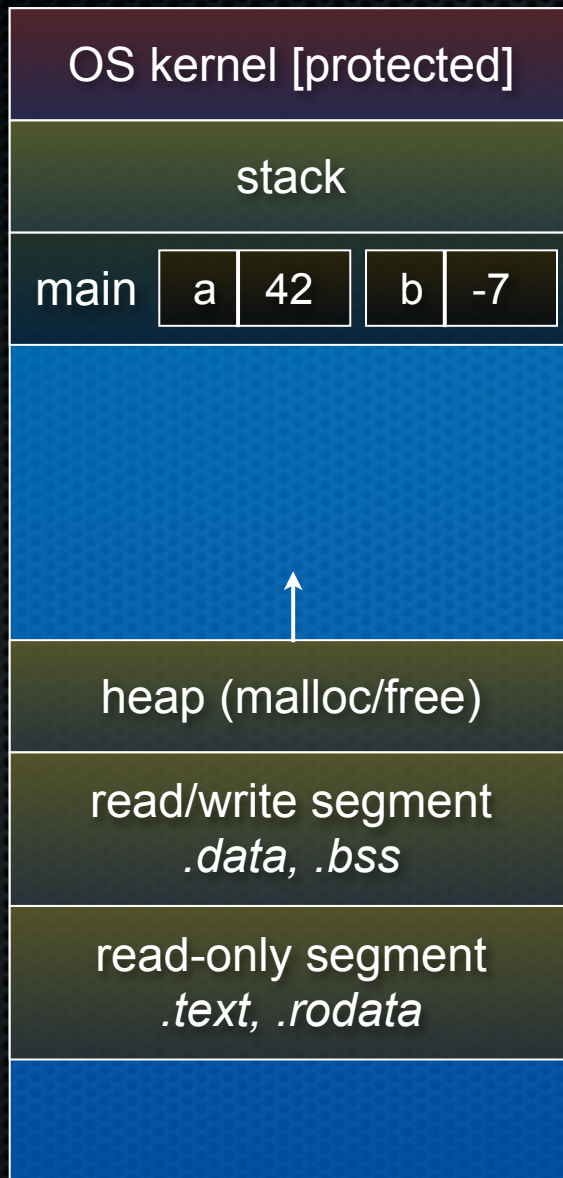
Pass-by-value (stack)



```
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
 - ▶ 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

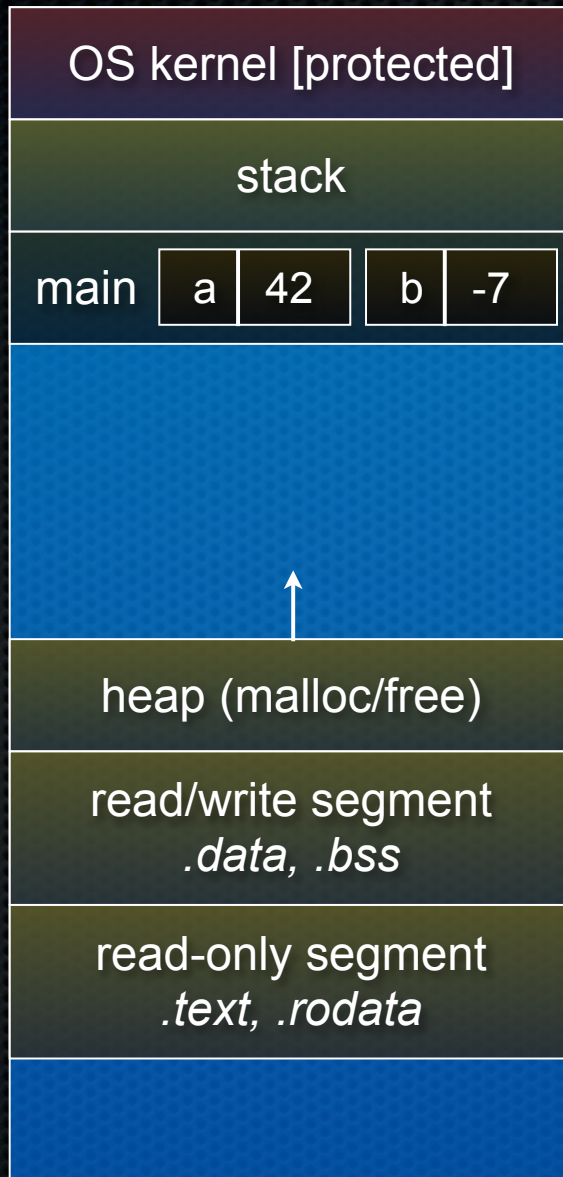
```
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

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

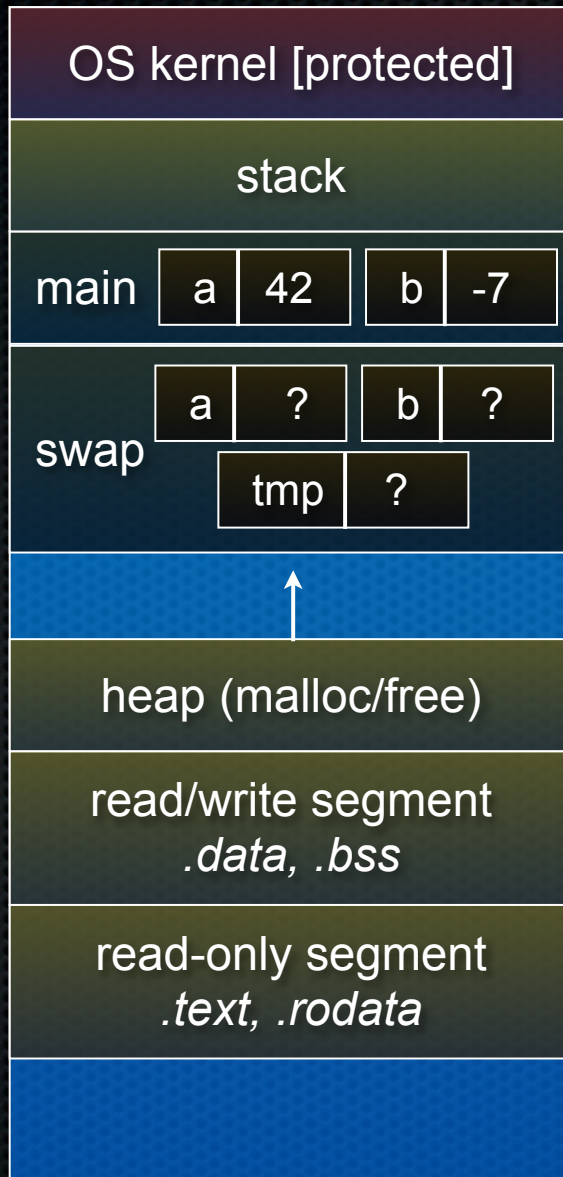
Pass-by-reference (stack)



```
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

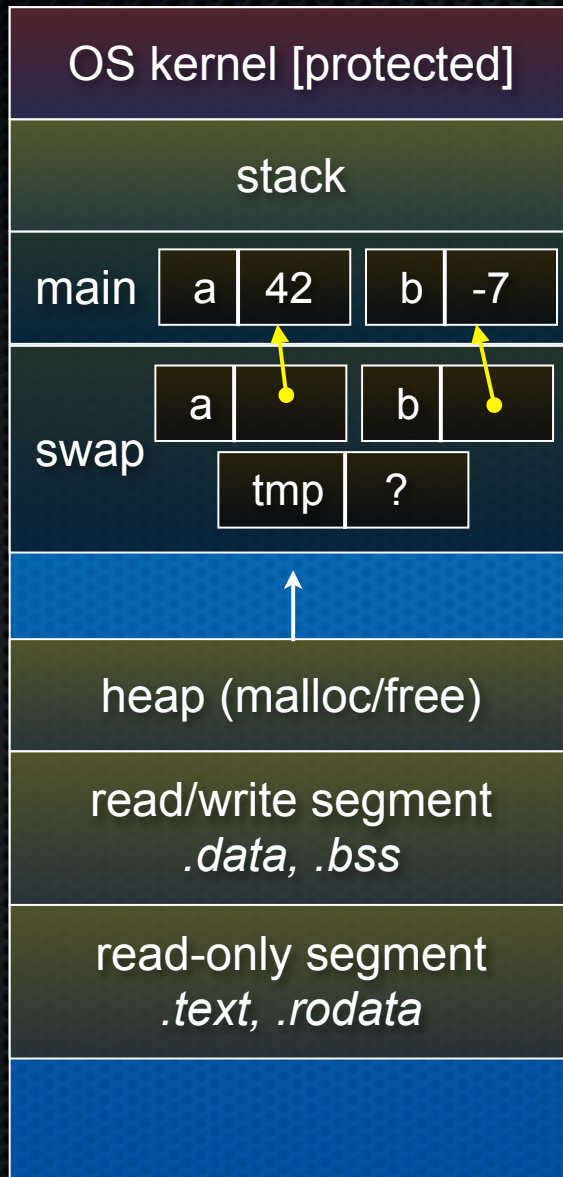
Pass-by-reference (stack)



```
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

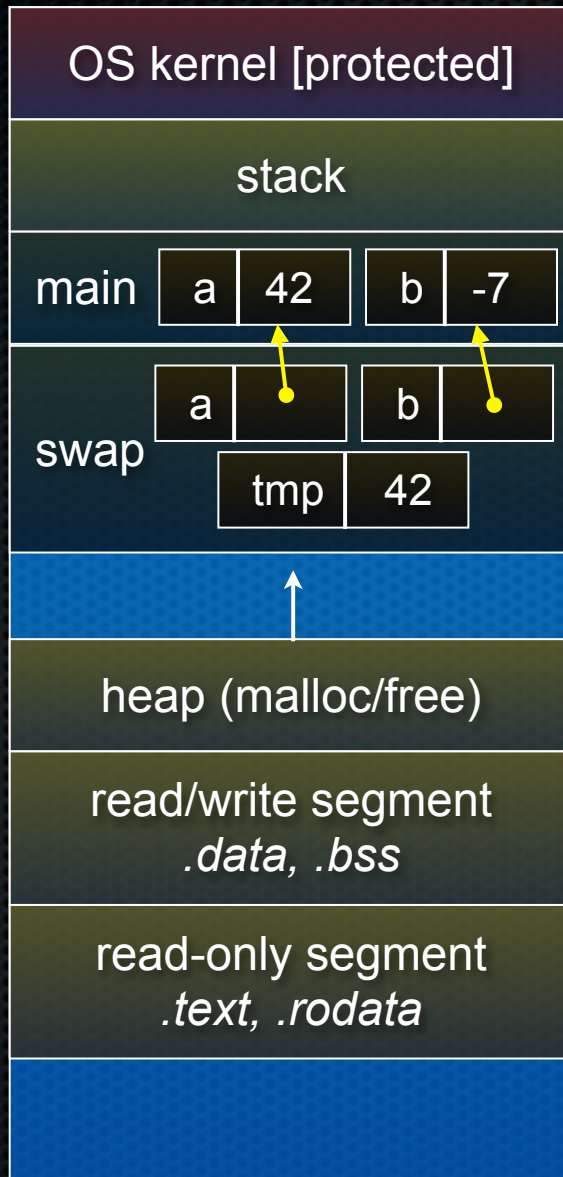
Pass-by-reference (stack)



```
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

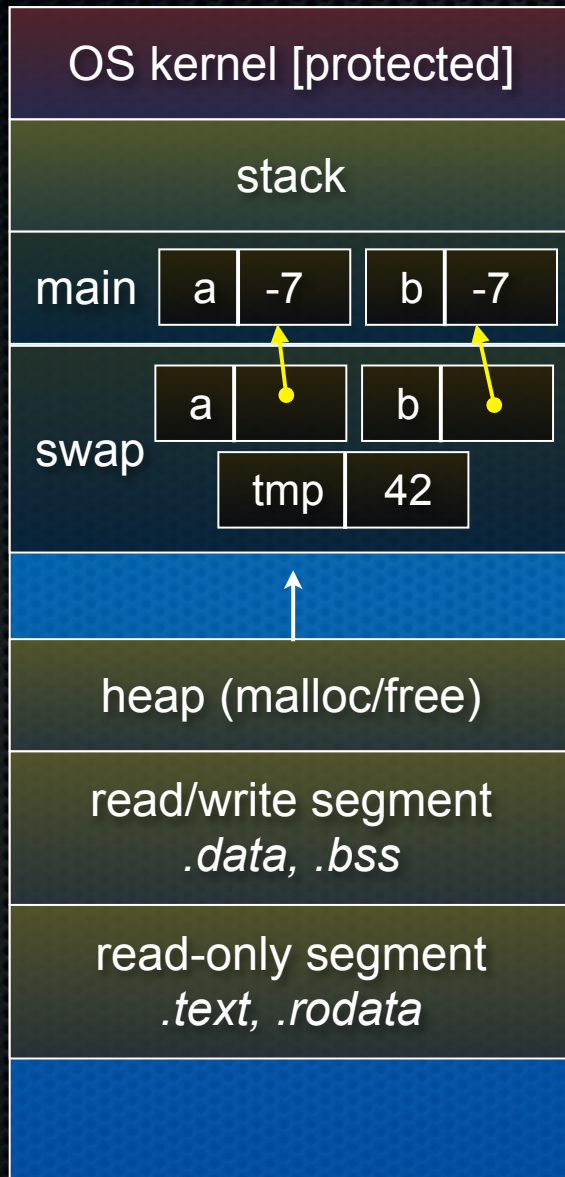
Pass-by-reference (stack)



```
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

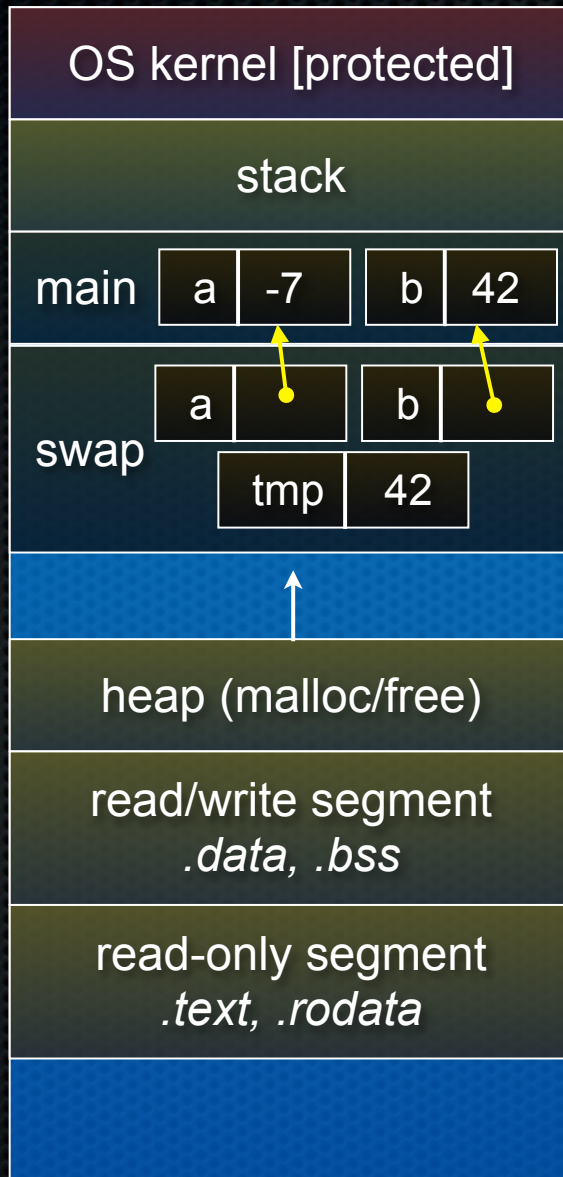
Pass-by-reference (stack)



```
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

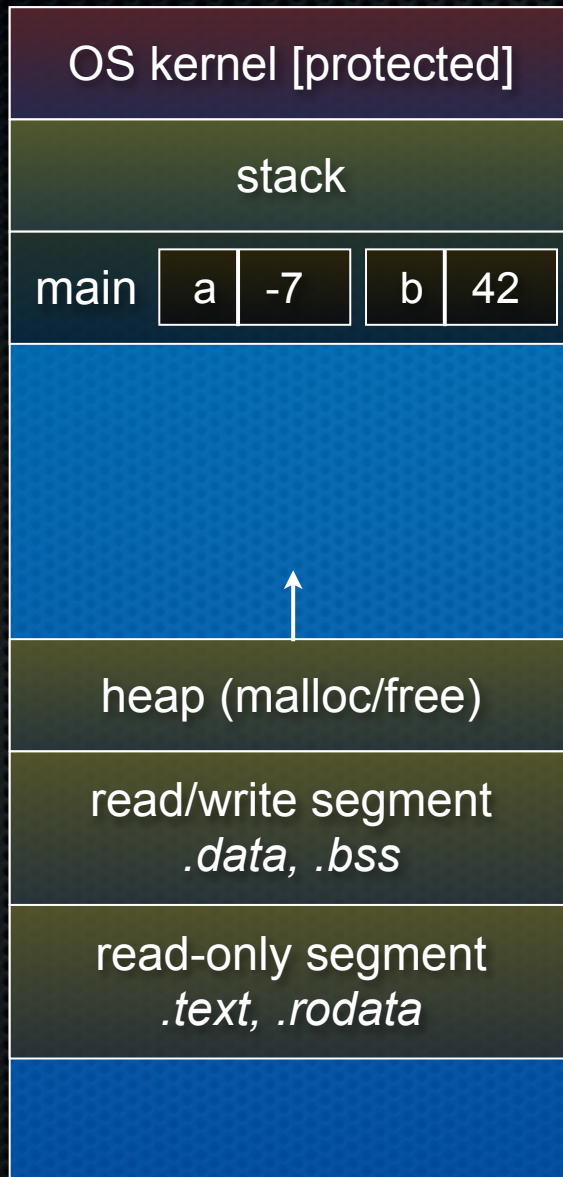
Pass-by-reference (stack)



```
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

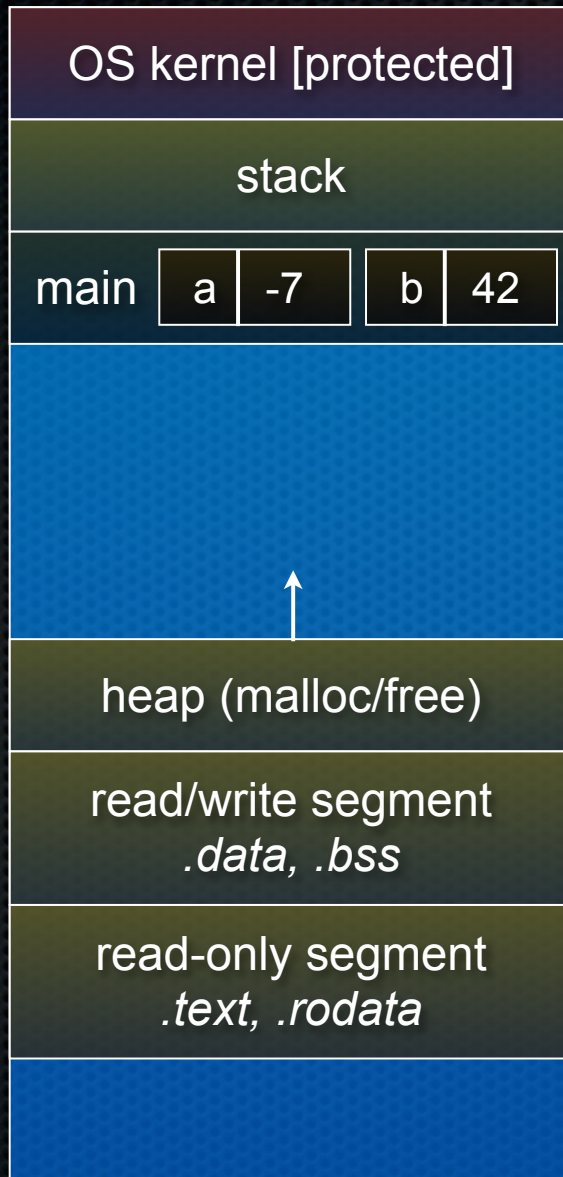
Pass-by-reference (stack)



```
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 on parameters 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 Wednesday!