Poll Everywhere

pollev.com/naomila

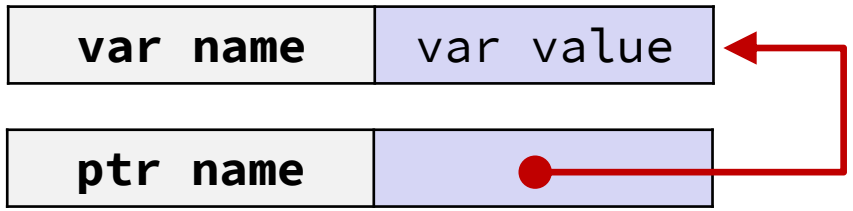
At this point in the code, what values are stored in `arr[]`?

A. {2, 3, 4}
B. {3, 4, 5}
C. {2, 6, 4}
D. {2, 4, 5}
E. girl idk...

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer
  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return EXIT_SUCCESS;
}
```

Hint! Use a box and arrow diagram:



Practice Solution (1/4)

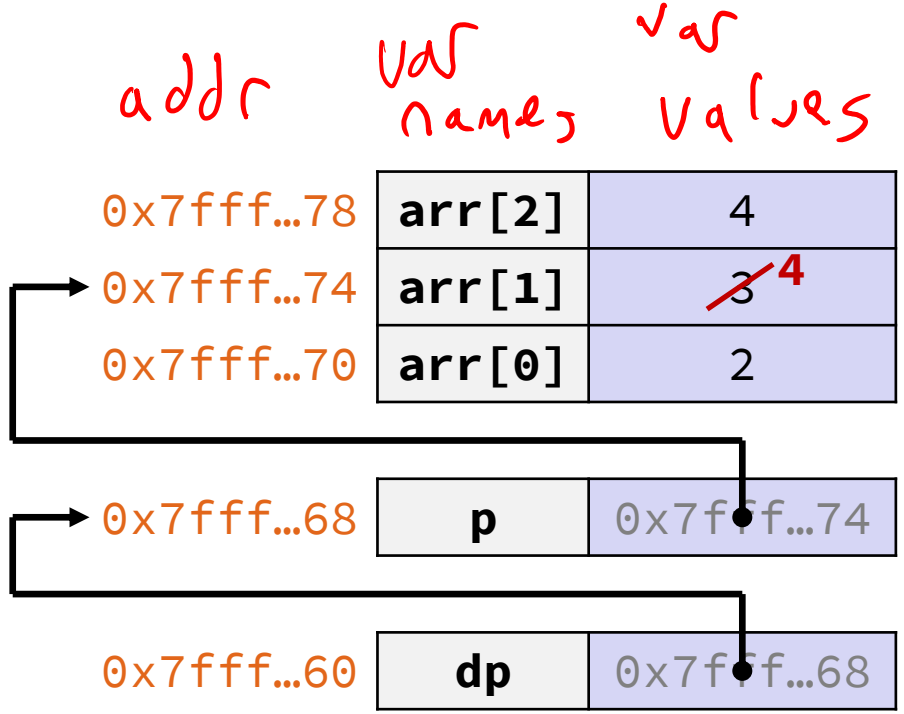
Note: arrow points to *next* instruction to be executed.

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return EXIT_SUCCESS;
}
```



address	name	value

Practice Solution (2/4)

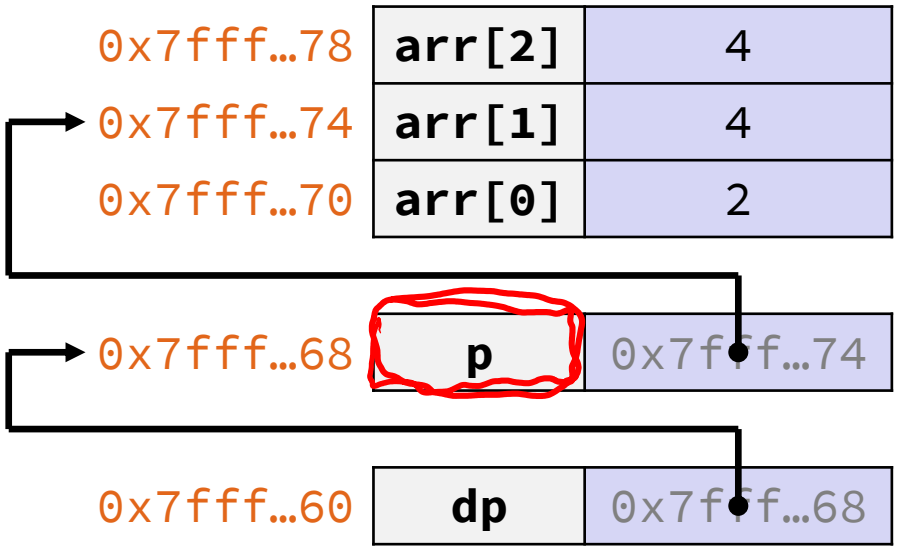
Note: arrow points to *next* instruction to be executed.

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return EXIT_SUCCESS;
}
```



address	name	value

Practice Solution (3/4)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return EXIT_SUCCESS;
}
```

address	name	value

Practice Solution (4/4)

{2, 4, 5}

Note: arrow points to *next* instruction to be executed.

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return EXIT_SUCCESS;
}
```

address	name	value

CSE333 Systems Programming

More Pointers

Instructors:

Naomi Alterman

Teaching Assistants:

Ann Baturytski	Barbora Buzkova	Mendel Carroll
Camden Harris	Hannah Hempstead	Rishabh Jain
Irene Lau	Jonathan Nister	Advay Patil
Aviel Wood	Angela Wu	Jennifer Xu

Relevant Course Information

- Exercise grades
  - We will occasionally give "Autograder Adjustment" points
  - Regrade requests for exercises: open 24 hr after, close 72 hr after release
- HW1 due next Thursday, 4/16 @ 11:59 PM
  - You ***may not*** modify interfaces (**.**h files), but ***do*** read the interfaces while you're implementing them (!)
  - Suggestion: pace yourself and make steady progress
- Gitlab repo usage
  - Commit things regularly (not all at once at the end)
  - Provides backups – can retrieve old versions of files ☺
  - Useful for sharing with staff members

Lecture Outline

- **Pointers as Parameters**
- Function Pointers

C is Call-By-Value

- C (and Java) pass arguments by *value*
  - Callee receives a **local copy** of the argument
    - Register or Stack
  - If the callee modifies a parameter, the 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);
  ...
```

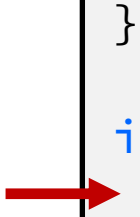
Broken Swap (1/7)

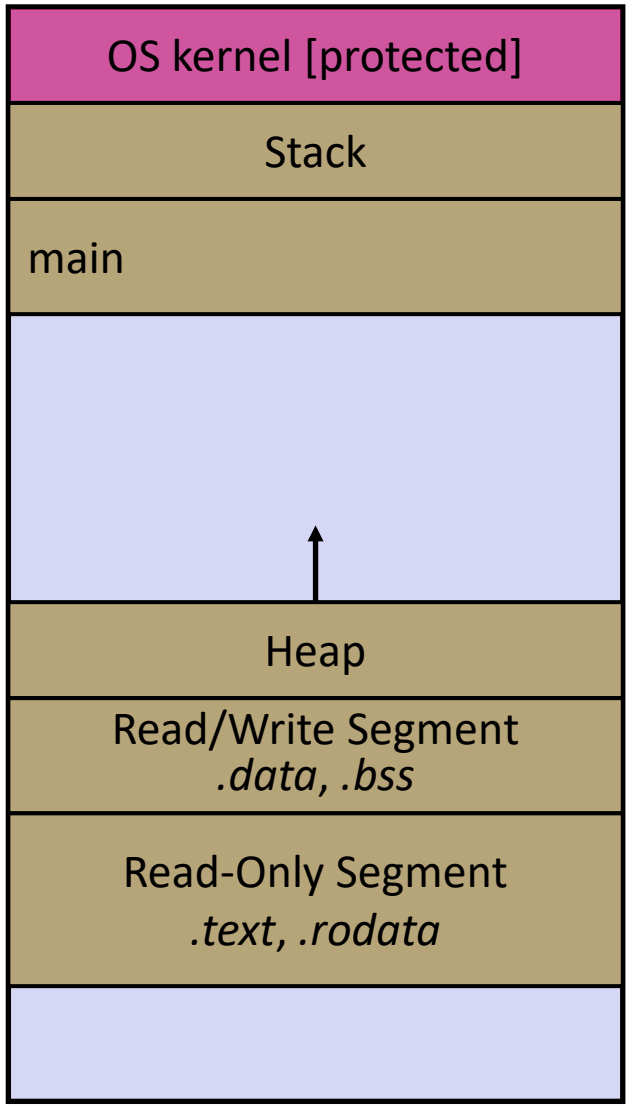
Note: Arrow points to *next* instruction.

brokenswap.c

```
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);
  ...
```



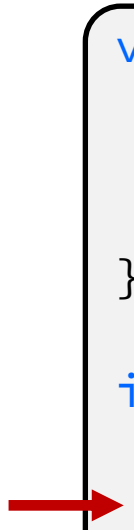


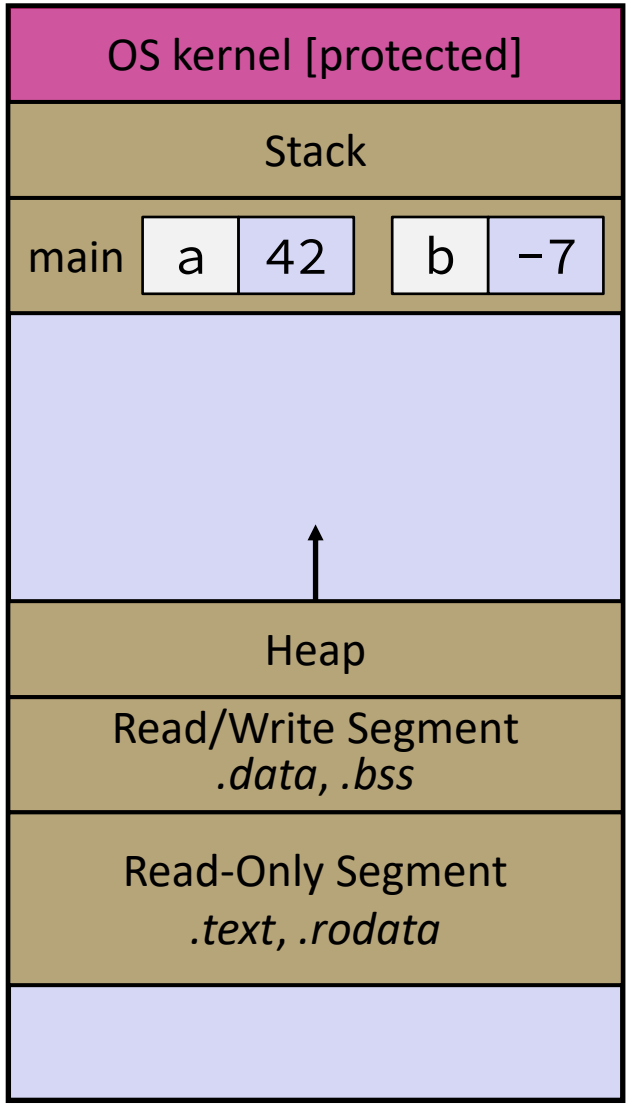
Broken Swap (2/7)

brokenswap.c

```
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);
  ...
```



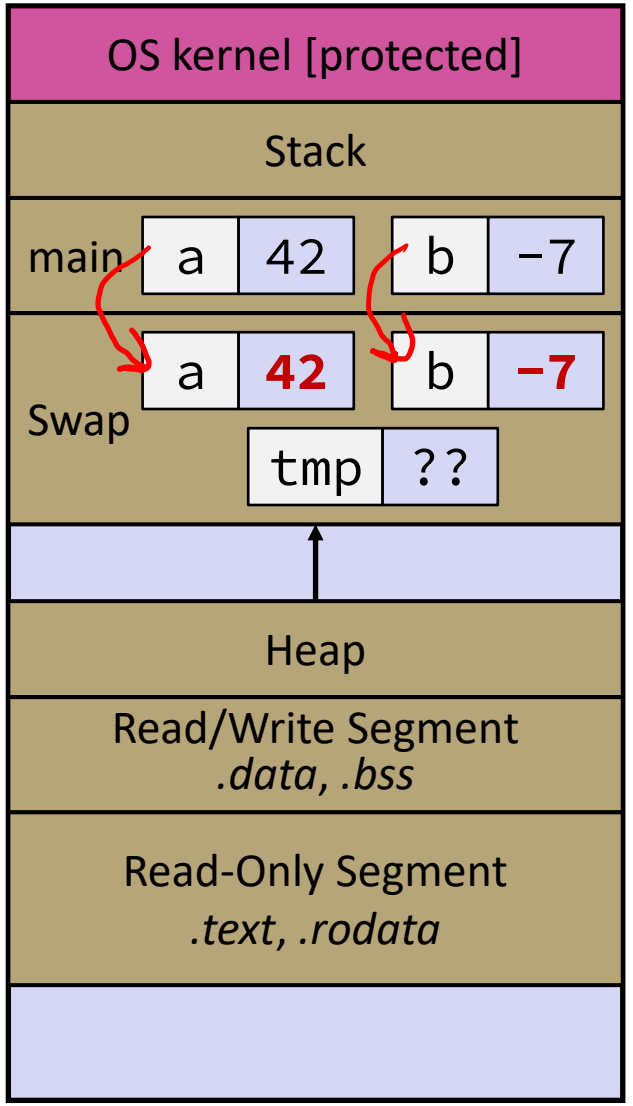


Broken Swap (3/7)

brokenswap.c

```
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);
  ...
```

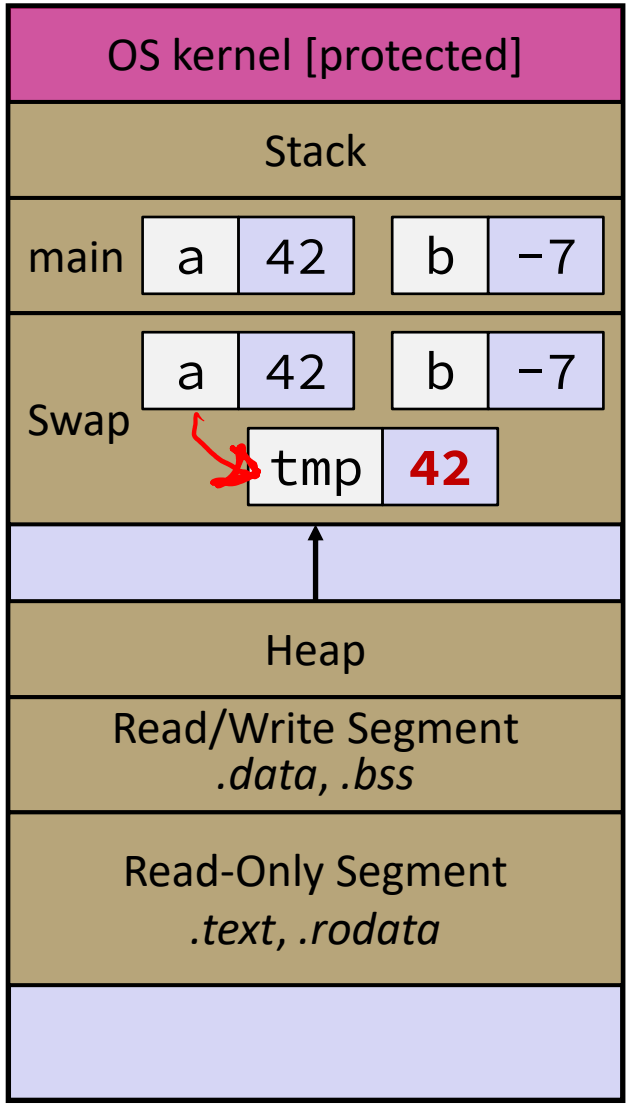


Broken Swap (4/7)

brokenswap.c

```
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);
  ...
```

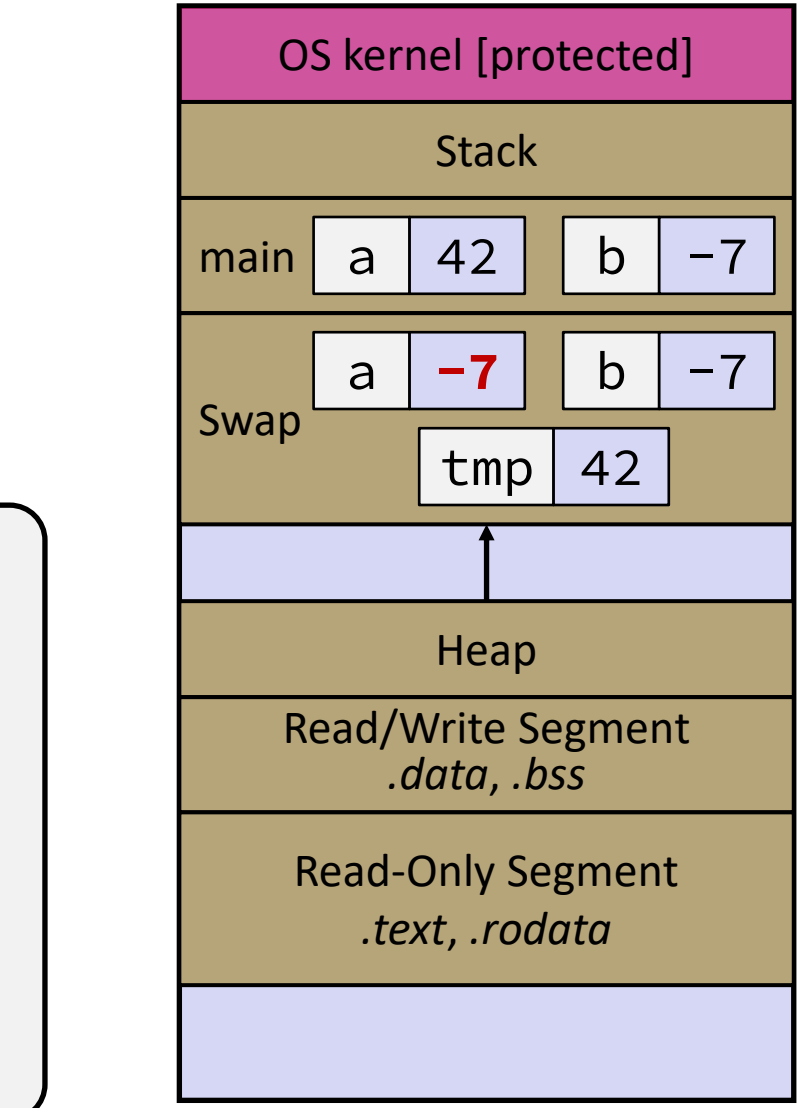


Broken Swap (5/7)

brokenswap.c

```
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);
  ...
```

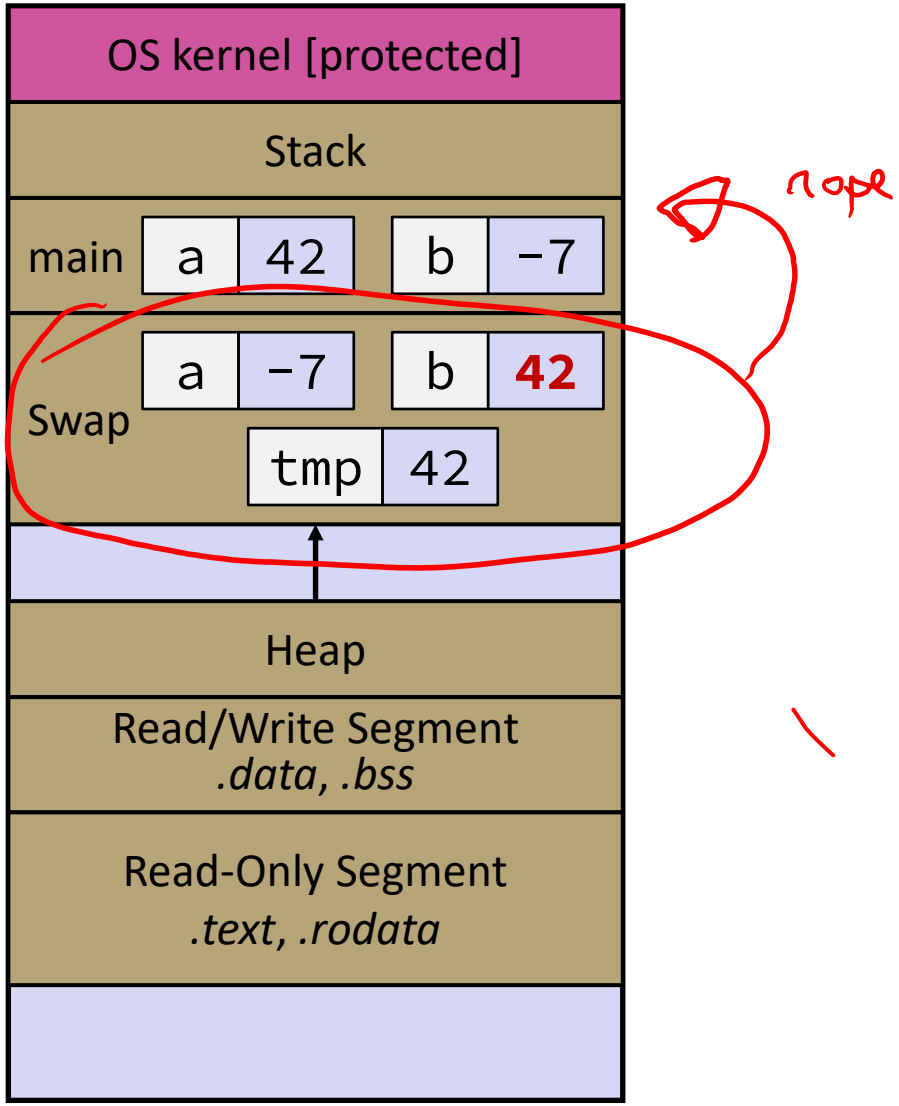


Broken Swap (6/7)

brokenswap.c

```
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);
  ...
```

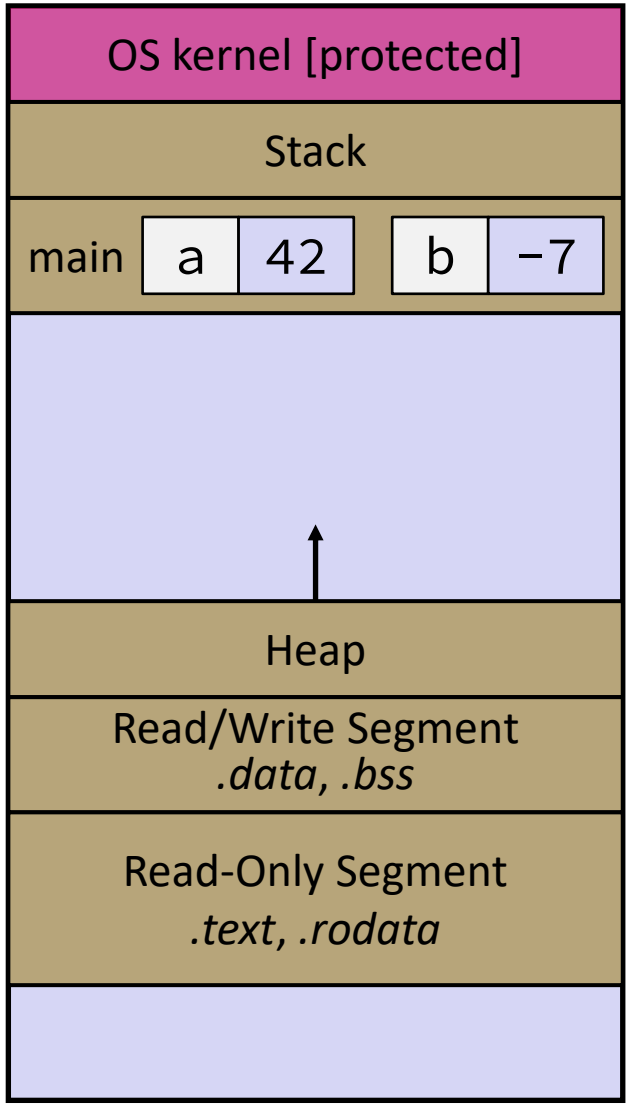


Broken Swap (7/7)

brokenswap.c

```
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);
  ...
```



Faking Call-By-Reference in C

- Can use pointers to *approximate* call-by-reference
  - Callee still receives a **copy** of the pointer (*i.e.*, call-by-value), but it can modify something in the caller's scope by dereferencing the pointer parameter

```
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);
  ...
```

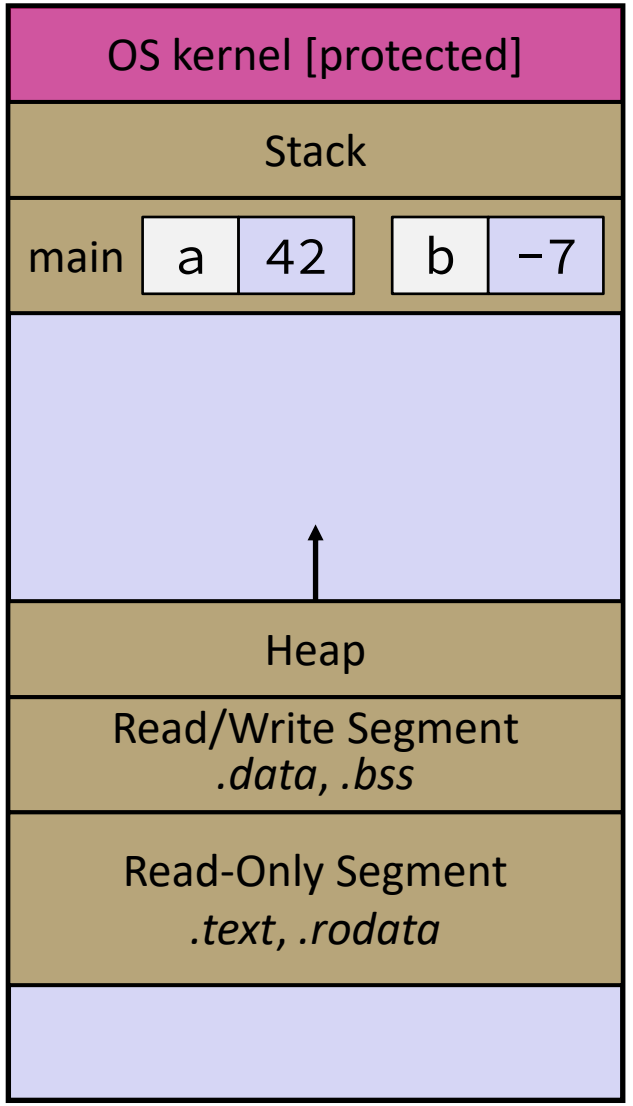
Fixed Swap (1/6)

Note: Arrow points to *next* instruction.

swap.c

```
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);
  ...
```

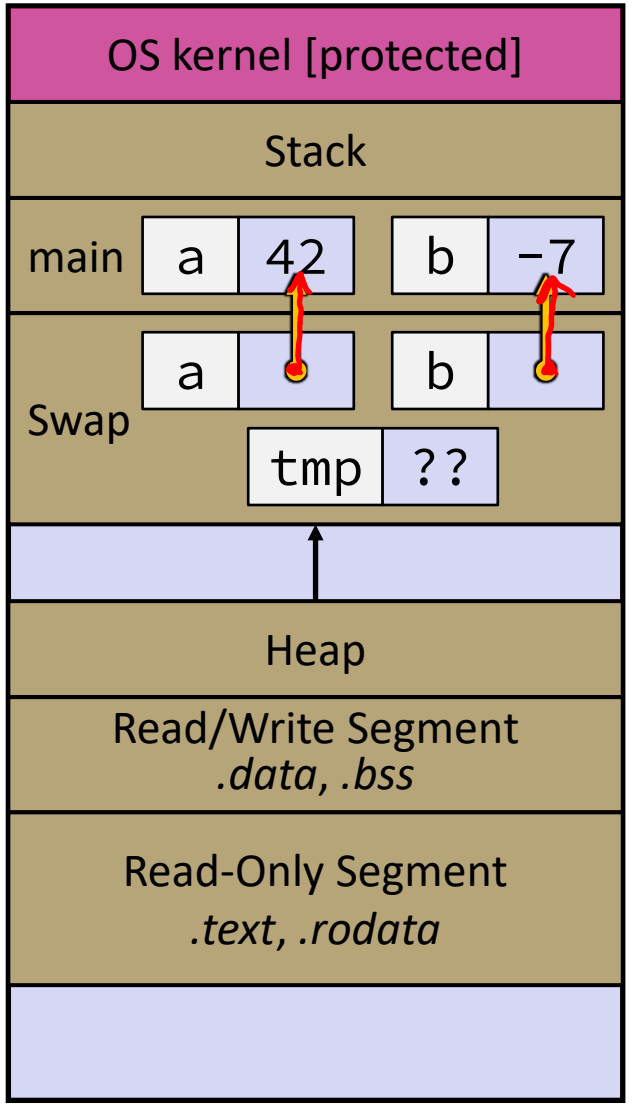


Fixed Swap (2/6)

swap.c

```
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);
  ...
```

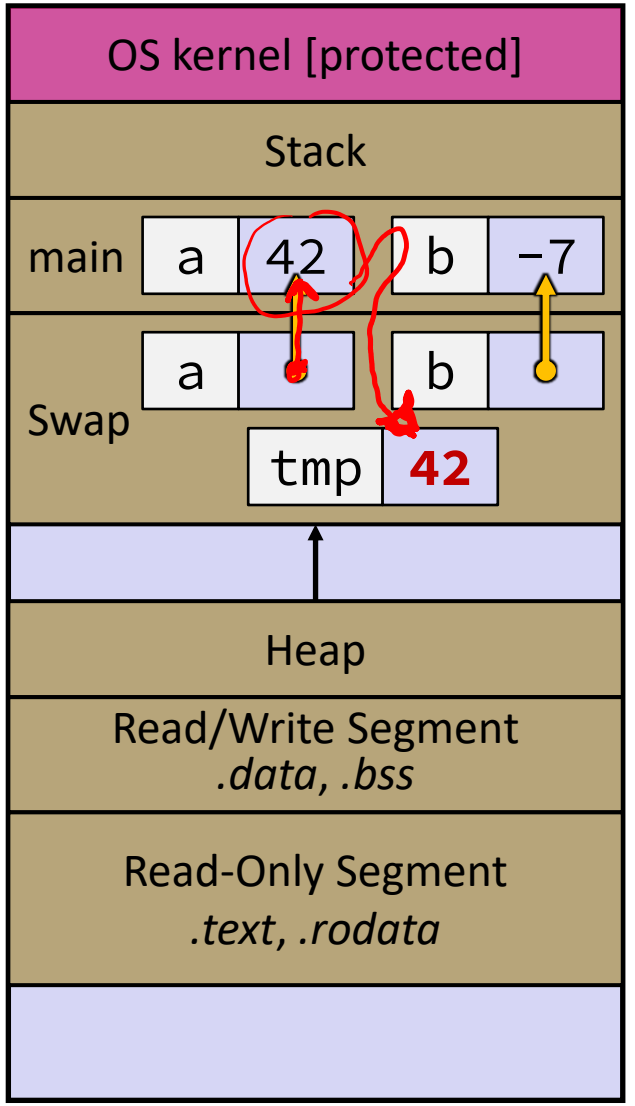


Fixed Swap (3/6)

swap.c

```
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);
  ...
```

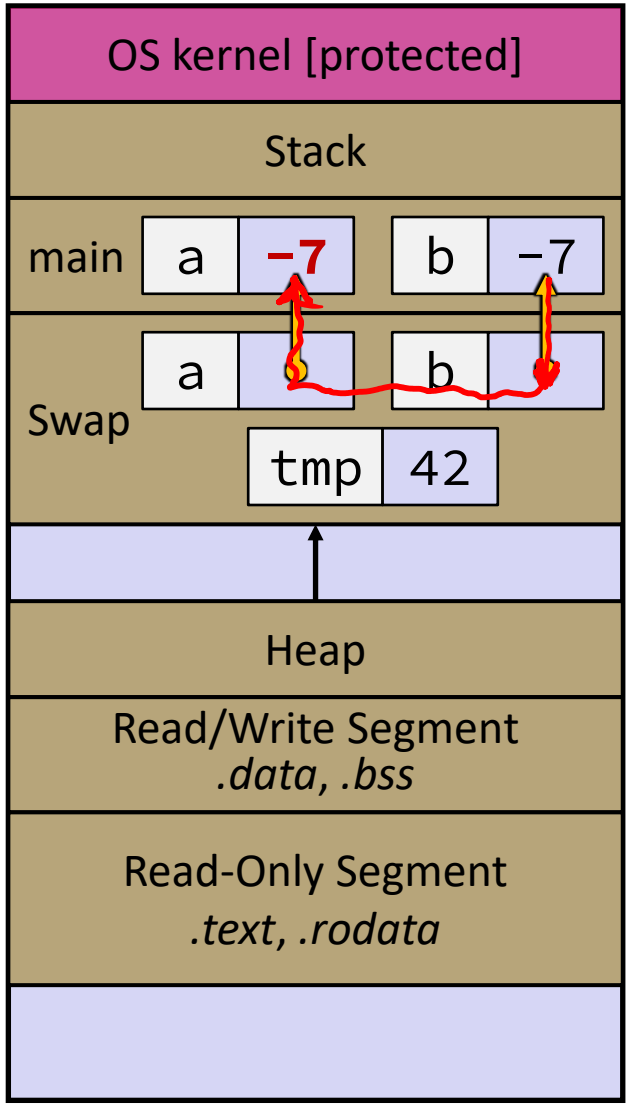


Fixed Swap (4/6)

swap.c

```
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);
  ...
```

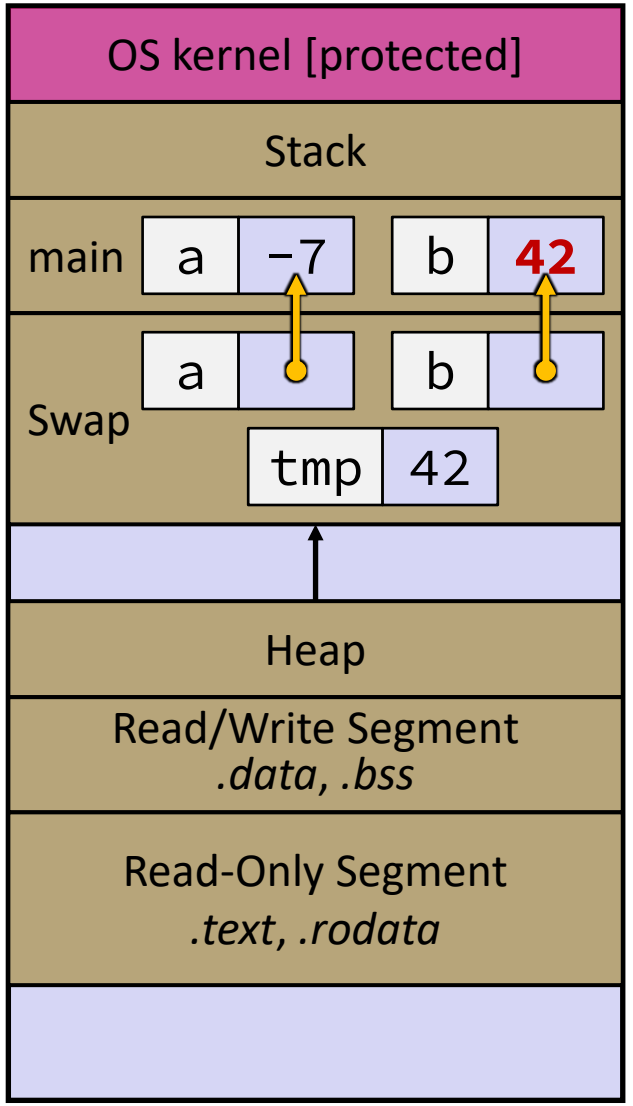


Fixed Swap (5/6)

swap.c

```
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);
  ...
```

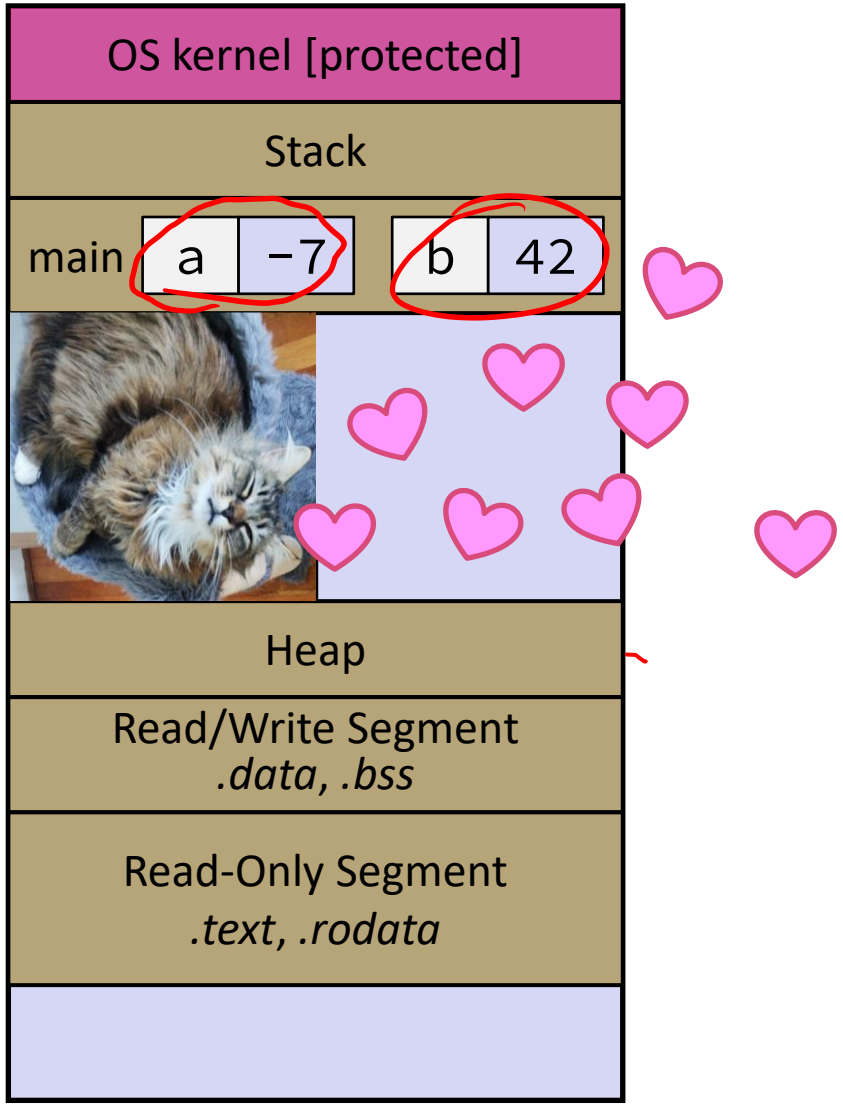


Fixed Swap (6/6)

swap.c

```
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);
  ...
```



Output Parameters

Warning: Misuse of output parameters is *the* largest cause of errors in this course!

- Output parameter
  - A pointer parameter used to store (via dereference) a function output *outside* of the function's stack frame
    - Typically points to/modifies something in the Caller's scope
  - Useful if you want to have multiple return values

- Setup and usage:
  1) Caller creates space for the data (*e.g.*, `type var;`)
  2) Caller passes in a pointer to Callee (*e.g.*, `&var`)
  3) Callee takes in output parameter (*e.g.*, `type* outparam`)
  4) Callee uses parameter to set output (*e.g.*, `*outparam = value;`)
  5) Caller accesses output via modified data (*e.g.*, `var`)

Poll Everywhere

pollev.com/naomila

Rank these ways of calling `GetGreeting()` from **best** to **worst**

…where **best** means the programmer's intent is clearest (and also isn't BUGGED OUT)

```
void GetGreeting(char** output) {
  *output = "Hello there\n";
}
```

A. 2

```
char** result;
GetGreeting(result);
printf("%s", *result);
```

B. 3

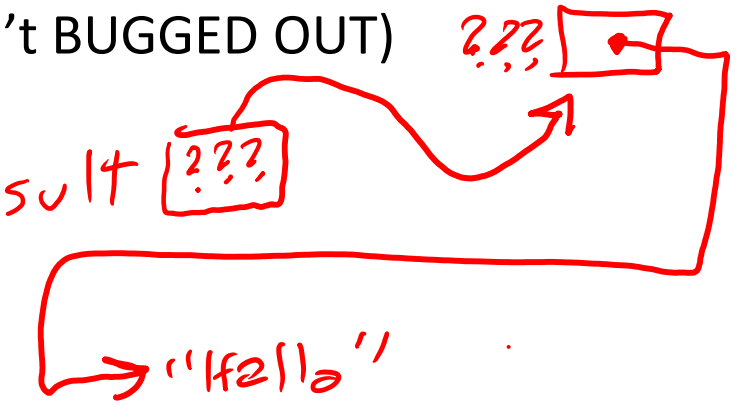
```
char* str;
char** result = &str;
GetGreeting(result);
printf("%s", str);
```

C. 4

```
char* result[1] = {NULL};
GetGreeting(result);
printf("%s", result[0]);
```

D. 1

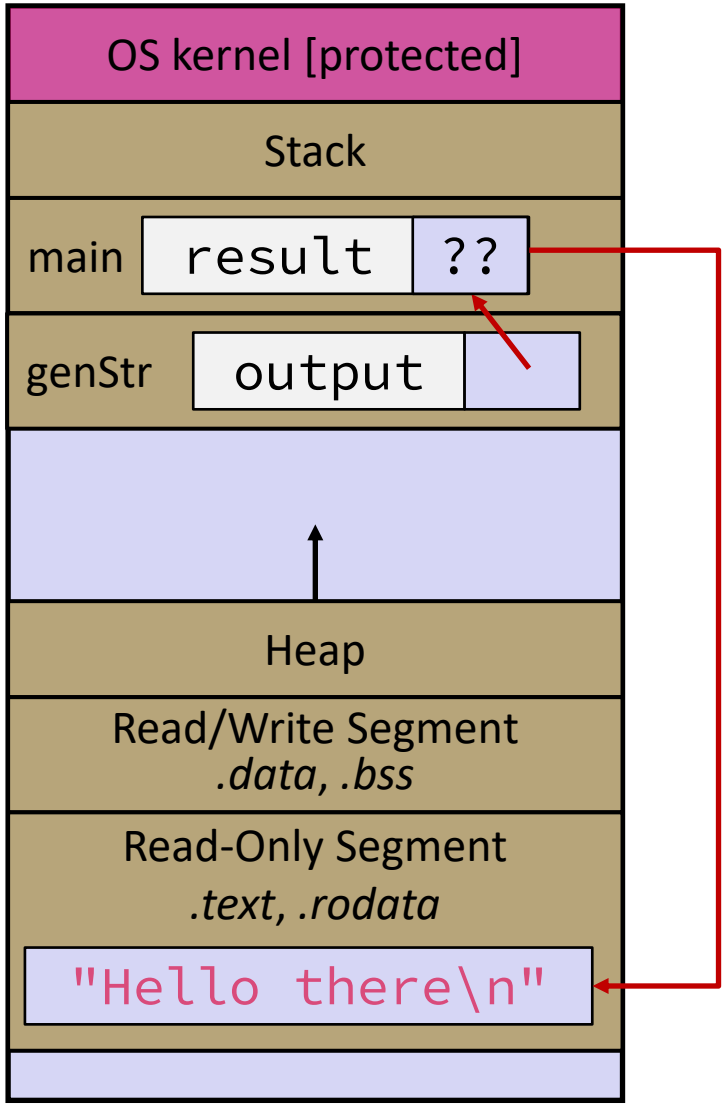
```
char* result;
GetGreeting(&result);
printf("%s", result);
```



Preferred Usage

Note: Arrow points to *next* instruction.

genstr.c



D.

```
void GenerateString(char** output);

int main(int argc, char** argv) {
  char* result;
  GenerateString(&result);
  printf("%s", result);

  return EXIT_SUCCESS;
}

void GenerateString(char** output) {
  *output = "Hello there\n";
}
```

- ✓ Works correctly (unlike A)
- ✓ Minimizes memory usage (unlike B)
- ✓ Intent is clear (unlike C)

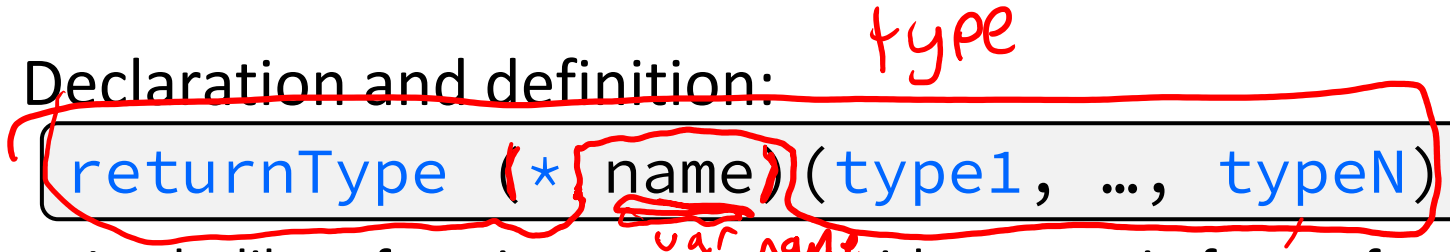
Lecture Outline

- Pointers as Parameters
- **Function Pointers**

Function Pointers

- Based on what you know about assembly, what is a function name, really?
  - Can use pointers that store addresses of functions!

- Declaration and definition:




```
returnType (* name)(type1, …, typeN) = dstFuncName
```

  - Looks like a function prototype with extra * in front of name
  - Set it *equal* to the location of the machine code for "dstFuncName"
  - Why are parentheses around (* name) needed?

- Using the function: `name(arg1, …, argN)`
  - Calls the pointed-to function with the given arguments and return the return value

Function Pointer Example

- Map() performs operation on each element of an array

```c
#define LEN 4

int Negate(int num) {return -num;}
int Square(int num) {return num * num;}

// perform operation pointed to on each array element
void Map(int a[], int len, int (* op)(int n)) {
  for (int i = 0; i < len; i++) {
    a[i] = (*op)(a[i]);  // dereference function pointer
  }
}
```

funcptr parameter

funcptr dereference

funcptr declaration

funcptr assignment

map.c

Function Pointer Example

- Map() performs operation on each element of an array

```
#define LEN 4

int Negate(int num) {return -num;}
int Square(int num) {return num * num;}

// perform operation pointed to on each array element
void Map(int a[], int len, int (* op)(int n)) {
  for (int i = 0; i < len; i++) {
    a[i] = (*op)(a[i]);  // dereference function pointer
  }
}

int main(int argc, char** argv) {
  int arr[LEN] = {-1, 0, 1, 2};
  int (* op)(int n);    // function pointer called 'op'
  op = Square;          // function name returns addr (like array)
  Map(arr, LEN, op);
  ...
```

funcptr parameter

funcptr dereference

funcptr declaration

funcptr assignment

map.c

Extra Exercise #1

- Write a program that defines:
  - A new structured type Point
    - Represent it with `floats` for the x and y coordinates
  - A new structured type Rectangle
    - Assume its sides are parallel to the x-axis and y-axis
    - Represent it with the bottom-left and top-right Points
  - A function that computes and returns the area of a Rectangle
  - A function that tests whether a Point is inside of a Rectangle

Extra Exercise #2

- Write a function that:
  - Accepts a function pointer and an integer as arguments
  - Invokes the pointed-to function with the integer as its argument