



CS107, Lecture 8

Introduction to Pointers

Reading: K&R (1.9, 5.5, Appendix B3) or Essential C section 3

Ed Discussion: <https://edstem.org/us/courses/65949/discussion/5447944>

Pointers

- A **pointer** is a variable that stores the address of some figure in memory.
- There is no true pass-by-reference in C like there is in C++, so we rely on pointers to share the addresses of variables with other functions.
- A single pointer can identify a single byte or an arbitrarily large data structure.
- Pointers are essential to dynamic memory allocation (coming soon).
- Pointers allow us to generically identify memory (coming less soon, but still soon).

Memory

- Memory is a big array of bytes.
- Each byte has a unique numeric index that is generally written in hexadecimal.
- A pointer stores any one of these memory addresses.

Address	Value
	...
0x105	'\0'
0x104	'e'
0x103	'l'
0x102	'p'
0x101	'p'
0x100	'a'
	...

Looking Back at C++

How would we write a program with a function that takes in an **int** and modifies it? We might use *pass by reference*.

```
void myFunc(int& num) {  
    num = 3;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 3!  
    ...  
}
```

Looking Ahead to C

- **All parameters** in C are passed "by value". For efficiency reasons, arrays (and strings, by extension) passed in as parameters are really caught as pointers.
- If an address is passed as a parameter, the address itself is copied as all parameters are. But because that address is the location of data meaningful to program execution, we have access to, and can even modify, that data.
- More generally, if we want to modify a parameter value in a function and have any changes persist afterward the function returns, we can share the location of the value—that is, share its address—instead of sharing the value itself. This way we copy the *address* instead of the *value*.

Pointers

```
int x = 2;
```

```
// Make a pointer that stores the address of x.
```

```
// (& means "address of")
```

```
int *xptr = &x;
```

If **declaration**: "pointer"
ex: `int *` is "pointer to an int"
*
If **operation**: "dereference/the value at address"
ex: `*num` is "the value at address num"

```
// Dereference the pointer to go to that address.
```

```
// (* means "dereference")
```

```
printf("%d", *xptr);    // prints 2
```

Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

Pointers

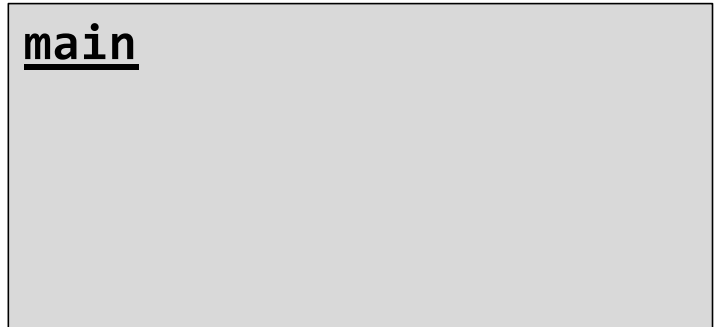
A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

STACK

main

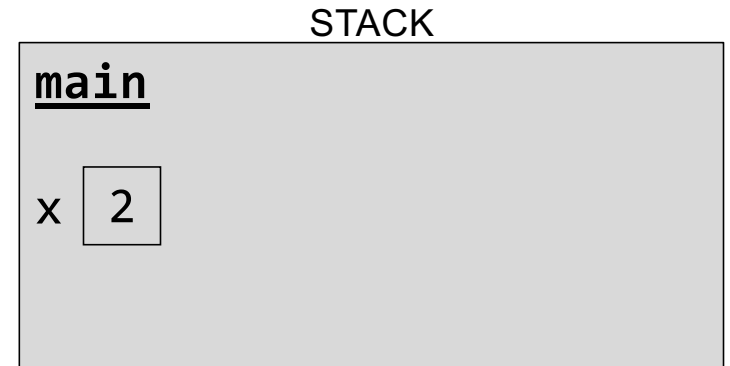


Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

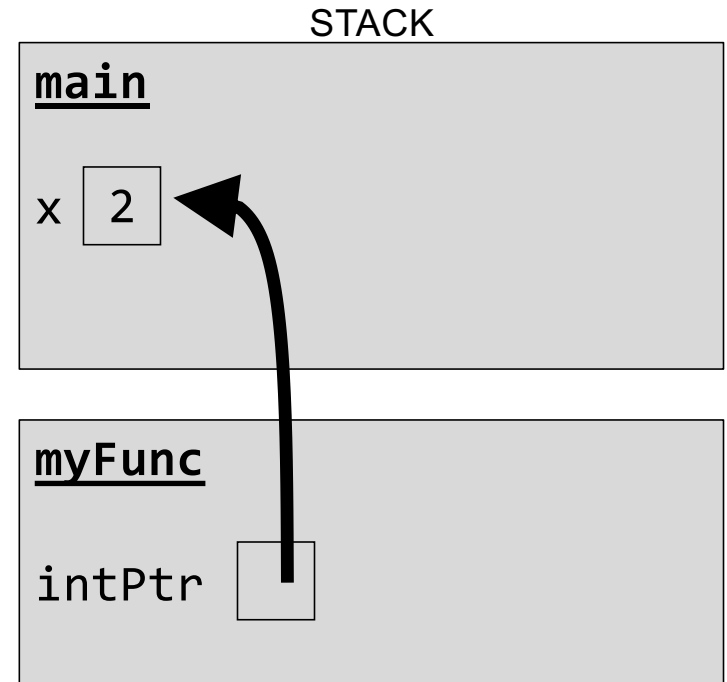
```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```



Pointers

A pointer is a variable that stores a memory address.

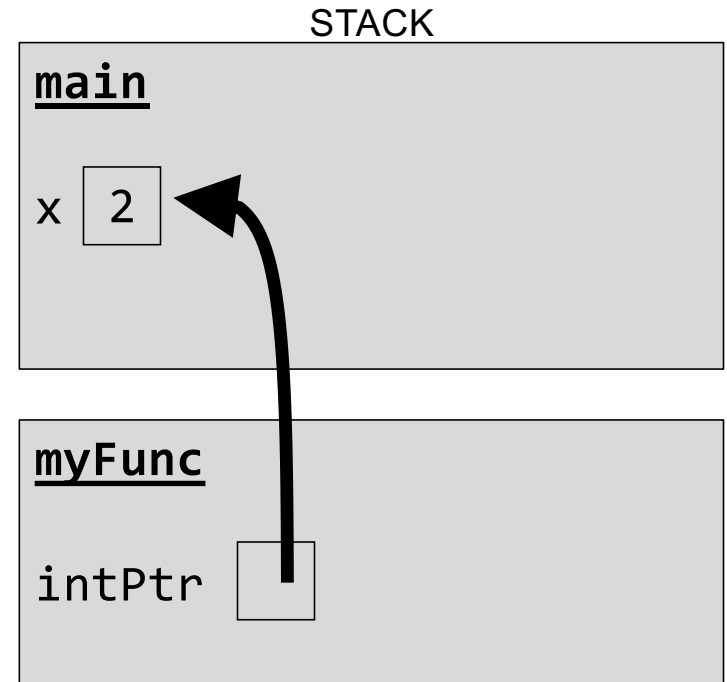
```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```



Pointers

A pointer is a variable that stores a memory address.

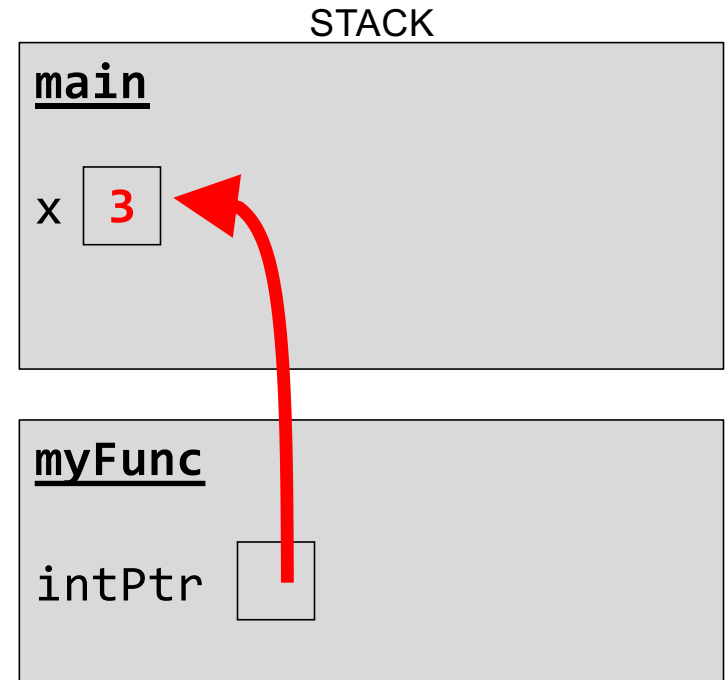
```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```



Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

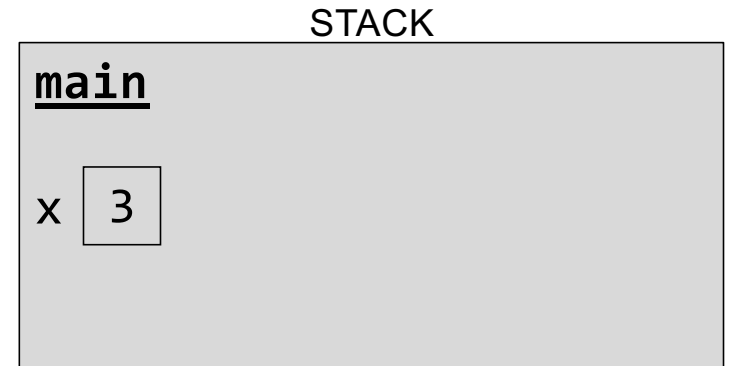


Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

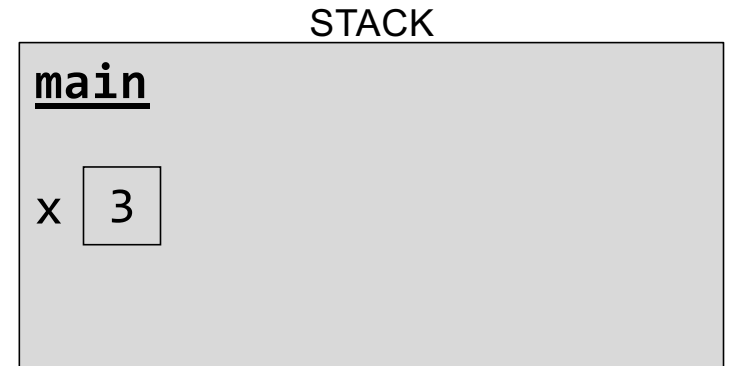


Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```



Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

main()



STACK		
Address		Value
		...
x	0x1f0	2
		...

Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

main()



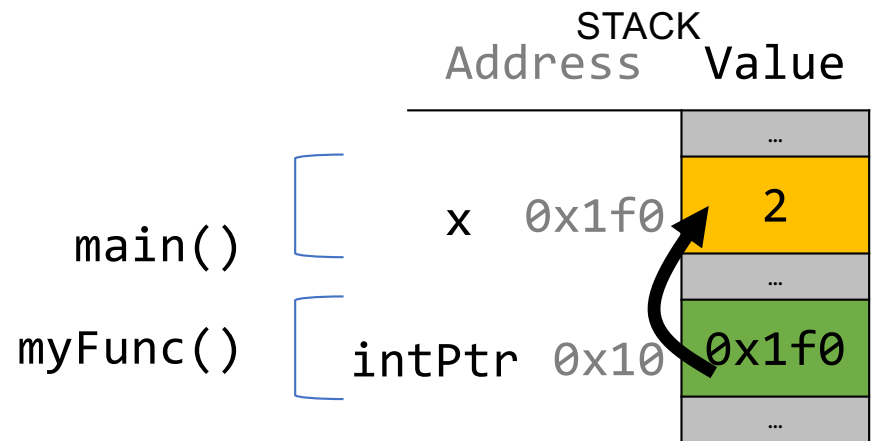
STACK		
Address		Value
		...
x	0x1f0	2
		...

Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

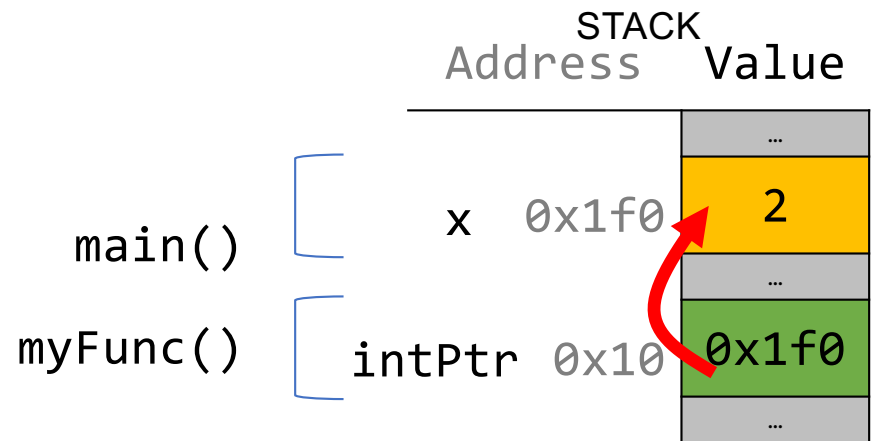


Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

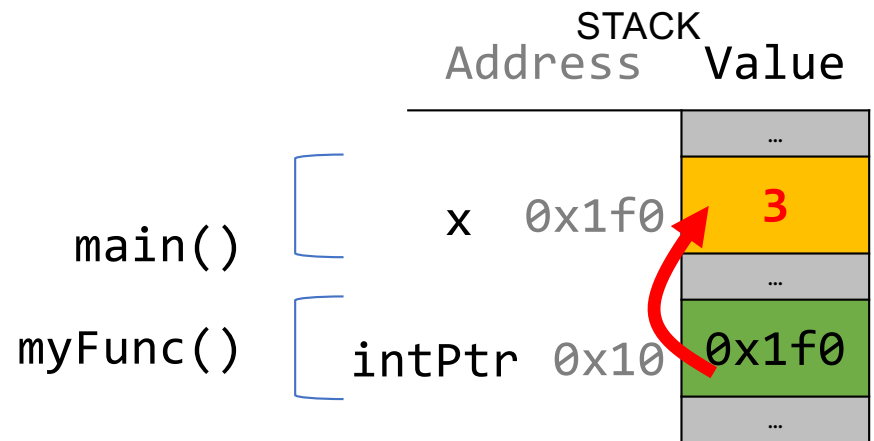


Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```



Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

main()



STACK	
Address	Value
x 0x1f0	...
	3
	...

Pointers

A pointer is a variable that stores a memory address.

```
void myFunc(int *intPtr) {  
    *intPtr = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(&x);  
    printf("%d", x);    // 3!  
    ...  
}
```

main()



STACK		
Address		Value
		...
x	0x1f0	3
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()



STACK		
Address		Value
		...
x	0x1f0	2
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()



STACK		
Address		Value
		...
x	0x1f0	2
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()

myFunc()

STACK
Address Value

		...
x	0x1f0	2
		...
val	0x10	2
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()

myFunc()

STACK
Address Value

		...
x	0x1f0	2
		...
val	0x10	2
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()

myFunc()

STACK
Address Value

		...
x	0x1f0	2
		...
val	0x10	3
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()



STACK		
Address		Value
x	0x1f0	...
		2
		...

Pointers

Without pointers, we would make copies.

```
void myFunc(int val) {  
    val = 3;  
}
```

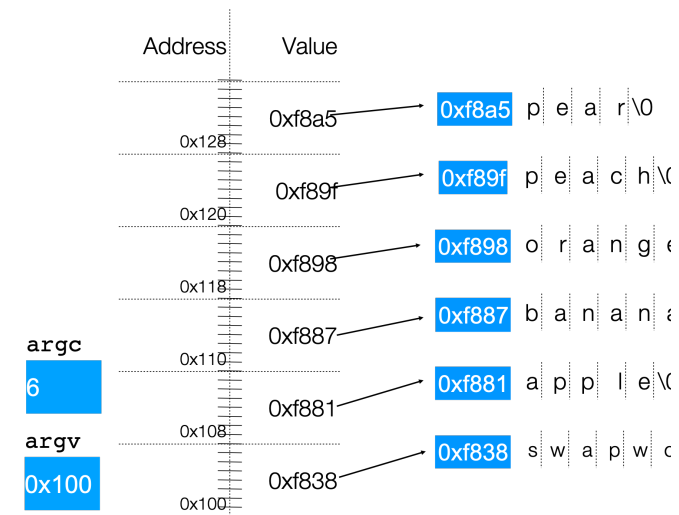
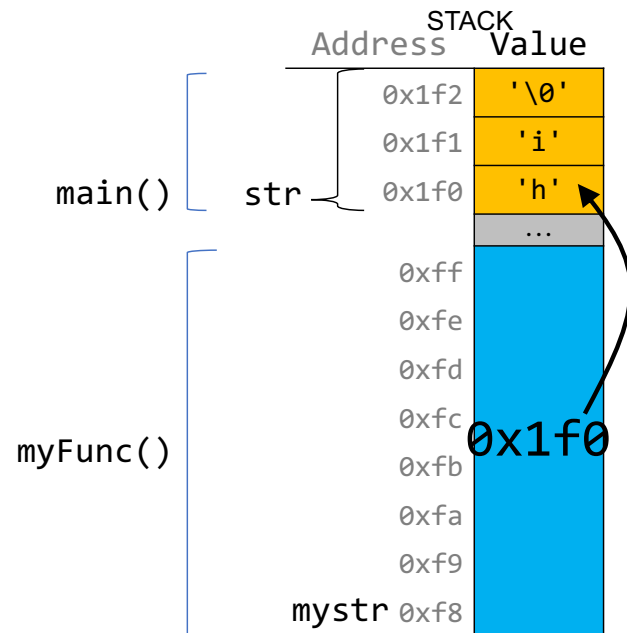
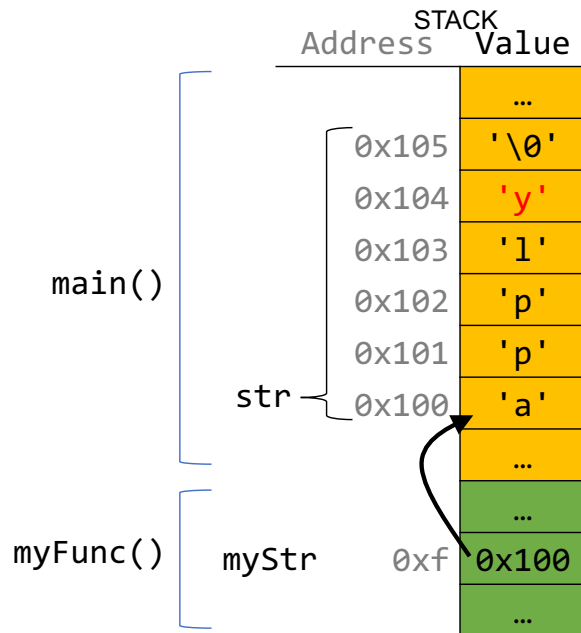
```
int main(int argc, char *argv[]) {  
    int x = 2;  
    myFunc(x);  
    printf("%d", x);    // 2!  
    ...  
}
```

main()



STACK		
Address		Value
		...
x	0x1f0	2
		...

How to draw memory diagrams?



Choose whatever style is convenient for you, keeping in mind that (1) memory is contiguous, and (2) C types are different sizes.

C Parameters

When you pass a value as a parameter, C passes a copy of that value.

```
void myFunction(int x) {  
    ...  
}  
  
int main(int argc, char *argv[]) {  
    int num = 4;  
    myFunction(num);           // passes copy of 4  
}
```

C Parameters

When you pass a value as a parameter, C passes a copy of that value.

```
void myFunction(int *x) {  
    ...  
}  
  
int main(int argc, char *argv[]) {  
    int num = 4;  
    myFunction(&num);           // passes copy of e.g. 0xffed63  
}
```

C Parameters

When you pass a value as a parameter, C passes a copy of that value.

```
void myFunction(char ch) {  
    ...  
}  
  
int main(int argc, char *argv[]) {  
    char *myStr = "Hello!";  
    myFunction(myStr[1]);           // passes copy of 'e'  
}
```

C Parameters

If you are modifying a specific instance of some value, pass the *location* of what you would like to modify.

Do I care about modifying *this* instance of my data? If so, I need to pass where that instance lives, as a parameter, so it can be modified.

Pointers

If you are modifying a specific instance of some value, pass the *location* of what you would like to modify.

```
void capitalize(char *ch) {  
    // modifies what is at the address stored in ch  
}  
  
int main(int argc, char *argv[]) {  
    char letter = 'h';  
    /* We don't want to capitalize any instance of 'h'.  
     * We want to capitalize *this* instance of 'h'! */  
    capitalize(&letter);  
    printf("%c", letter);    // want to print 'H';  
}
```

Pointers

If you are modifying a specific instance of some value, pass the *location* of what you would like to modify.

```
void doubleNum(int *x) {  
    // modifies what is at the address stored in x  
}  
  
int main(int argc, char *argv[]) {  
    int num = 2;  
    /* We don't want to double any instance of 2.  
     * We want to double *this* instance of 2! */  
    doubleNum(&num);  
    printf("%d", num); // want to print 4;  
}
```

Pointers

If a function takes an address (pointer) as a parameter, it can *go to* that address if it needs the actual value.

```
void capitalize(char *ch) {  
    // *ch gets the character stored at address ch.  
    char newChar = toupper(*ch);  
  
    // *ch = goes to address ch and puts newChar there.  
    *ch = newChar;  
}
```

Pointers Summary

- If you are performing an operation with some input and do not care about any changes to the input, **pass the data type itself**.
- If you are modifying a specific instance of some value, **pass the location** of what you would like to modify.
- If a function takes an address (pointer) as a parameter, it can *go to* that address if it needs the actual value.
- If a function accepts an **int ***, it can modify the **int** at the supplied address.
- If a function accepts a **char ***, it can modify the **char** at the supplied address.
- If a function accepts an **char ****, it can modify the **char *** at the supplied address.