

CS107, Lecture 11

C Generics – Void *

Reading: None 😊

Ed Discussion: <https://edstem.org/us/courses/65949/discussion/5505467>

CS107 Topic 4: How can we use our knowledge of memory and data representation to write code that works with any data type?

CS107 Topic 4

How can we use our knowledge of memory and data representation to write code that works with any data type?

Why is answering this question important?

- Writing code that works with any data type lets us write more generic, reusable code while understanding potential pitfalls (today and Friday)
- Allows us to learn how to pass functions as parameters, a core concept in many languages (Friday and next Monday)

assign4: implement your own version of the **ls** command, a function to generically find and insert elements into a sorted array, and a program using that function to sort the lines in a file like the **sort** command.

Learning Goals

- Learn how to write C code that works with any data type.
- Learn about how to use void * and overcome its shortcomings.
- Learn about the potential harm from vulnerabilities, challenges to proper disclosure of vulnerabilities, and how we weigh competing interests.

Generics

- We generally strive to write code that is as general-purpose as possible.
- Generic code minimizes code duplication so that optimizations and bug fixes can be managed in one place instead of many.
- Generics are used throughout C to sort arrays of any type, search arrays of any type, free arbitrary memory, and so forth.
- How can we write generic code in C?

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

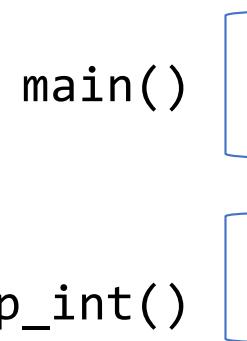
```
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

| Address | Stack Value |
|---------|-------------|
| x | 0xff14 2 |
| y | 0xff10 5 |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```



| Address | Stack Value |
|---------|--------------|
| x | 0xff14 2 |
| y | 0xff10 5 |
| b | 0xf18 0xff10 |
| a | 0xf10 0xff14 |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

| | Address | Stack Value |
|------|---------|-------------|
| x | 0xff14 | 2 |
| y | 0xff10 | 5 |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff14 |
| temp | 0xf0c | 2 |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

main()

swap_int()

| Address | Stack Value |
|---------|--------------|
| x | 0xff14 5 |
| y | 0xff10 5 |
| b | 0xf18 0xff10 |
| a | 0xf10 0xff14 |
| temp | 0xf0c 2 |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

main()

swap_int()

| Address | Stack Value |
|---------|--------------|
| x | 0xff14 5 |
| y | 0xff10 2 |
| b | 0xf18 0xff10 |
| a | 0xf10 0xff14 |
| temp | 0xf0c 2 |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

main()

| Address | Stack Value |
|---------|-------------|
| x | 0xff14 |
| y | 0xff10 |
| | ... |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

main()



| Address | Stack Value |
|---------|-------------|
| x | 0xff14 |
| y | 0xff10 |
| | ... |

Swap

You're asked to write a function that swaps two numbers.

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    int x = 2;  
    int y = 5;  
    swap_int(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

main()

| Address | Stack Value |
|---------|-------------|
| x | 0xff14 |
| y | 0xff10 |
| | ... |

**"Oh, when I said 'numbers'
I meant shorts, not ints."**



Swap

```
void swap_short(short *a, short *b) {
    short temp = *a;
    *a = *b;
    *b = temp;
}

int main(int argc, char *argv[]) {
    short x = 2;
    short y = 5;
    swap_short(&x, &y);
    // want x = 5, y = 2
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

Swap

```
void swap_short(short *a, short *b) {  
    short temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    short x = 2;  
    short y = 5;  
    swap_short(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %d, y = %d\n", x, y);  
    return 0;  
}
```

| | Address | Stack Value |
|------|---------|-------------|
| x | 0xff12 | 2 |
| y | 0xff10 | 5 |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff12 |
| temp | 0xf0e | 2 |

"You know what, I messed up. We're going to use strings. Could you write something to swap those?"



Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|--------------|---------|--------|
| | x | 0xff18 |
| main() | y | 0xff10 |
| | 0xf | '\0' |
| DATA SEGMENT | 0xe | '5' |
| | 0xd | '\0' |
| | 0xc | '2' |
| | | ... |

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|-----|---------|--------|
| x | 0xff18 | 0xc |
| y | 0xff10 | 0xe |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff18 |
| | ... | |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |
| | ... | |

main() []

swap_string() []

DATA SEGMENT []

Address 0xf18 Value 0xc

Address 0xf10 Value 0xe

Address 0xf18 Value 0xc

Address 0xf10 Value 0xe

Address 0xf Value '\0'

Address 0xe Value '5'

Address 0xd Value '\0'

Address 0xc Value '2'

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|------|---------|--------|
| x | 0xff18 | 0xc |
| y | 0xff10 | 0xe |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff18 |
| temp | 0xf08 | 0xc |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |

main() swap_string()

DATA SEGMENT

22

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|------|---------|--------|
| x | 0xff18 | 0xe |
| y | 0xff10 | 0xe |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff18 |
| temp | 0xf08 | 0xc |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |

main() []

swap_string() []

DATA SEGMENT []

23

The diagram illustrates the state of memory during the execution of the swap operation. The DATA SEGMENT contains the strings "2" and "5". In the main() stack frame, the local variables x and y point to the first byte of "5" and "2" respectively. In the swap_string() stack frame, the parameters a and b point to the second byte of "5" and "2" respectively. The local variable temp points to the third byte of "2". Red arrows indicate the movement of bytes between memory and registers.

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|------|---------|--------|
| x | 0xff18 | 0xe |
| y | 0xff10 | 0xc |
| b | 0xf18 | 0xff10 |
| a | 0xf10 | 0xff18 |
| temp | 0xf08 | 0xc |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |
| | | ... |

main() []

swap_string() []

DATA SEGMENT []

24

The diagram illustrates the state of memory during the execution of the swap operation. It shows three main sections: the stack frames for `main()` and `swap_string()`, and a separate `DATA SEGMENT`.

- Stack Frame for `main()`:** Contains local variables `x` (address 0xff18, value 0xe) and `y` (address 0xff10, value 0xc).
- Stack Frame for `swap_string()`:** Contains parameters `a` (address 0xf10, value 0xff18) and `b` (address 0xf18, value 0xff10), and a local variable `temp` (address 0xf08, value 0xc).
- DATA SEGMENT:** Contains four characters: '\0', '5', '\0', and '2' at addresses 0xf, 0xe, 0xd, and 0xc respectively.

Red arrows indicate the movement of data:

- From `x` (0xe) to the first character ('5').
- From `y` (0xc) to the second character ('\0').
- From `a` (0xff18) to the third character ('\0').
- From `b` (0xff10) to the fourth character ('2').

This visualizes how the swap operation swaps the pointers between the stack frames, leaving the original data values in the DATA SEGMENT unchanged.

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|-----|---------|-------|
| x | 0xff18 | ... |
| | 0xe | 0xe |
| y | 0xff10 | ... |
| | 0xc | 0xc |
| | ... | ... |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |
| | ... | ... |

main()

DATA SEGMENT

The diagram illustrates the state of memory after the swap operation. The 'main()' stack frame is shown with a blue bracket, and the 'DATA SEGMENT' is shown with a blue bracket. Red arrows point from the variable names 'x' and 'y' to their corresponding memory locations. The memory dump shows the values 0xe and 0xc at addresses 0xff18 and 0xff10 respectively, indicating that the pointers have been swapped but the original string data remains unchanged.

Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|-----|---------|-------|
| x | 0xff18 | ... |
| | 0xe | 0xe |
| y | 0xff10 | ... |
| | 0xc | 0xc |
| | ... | ... |
| 0xf | | '\0' |
| 0xe | | '5' |
| 0xd | | '\0' |
| 0xc | | '2' |
| | ... | ... |

main()

DATA SEGMENT

The diagram illustrates the state of memory after the swap operation. The 'main()' stack frame is shown with variables 'x' and 'y' pointing to memory locations 0xff18 and 0xff10 respectively. The DATA SEGMENT contains the string "5\02\0". Red arrows highlight the values at these addresses: '5' is at 0xff10 and '\0' is at 0xff18, indicating they have swapped places.

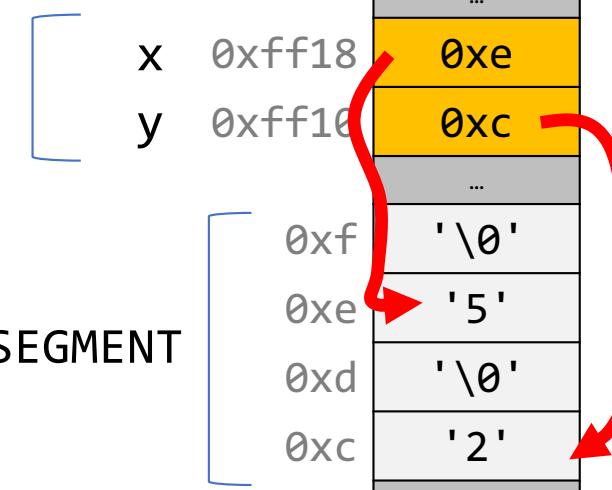
Swap

```
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
int main(int argc, char *argv[]) {  
    char *x = "2";  
    char *y = "5";  
    swap_string(&x, &y);  
    // want x = 5, y = 2  
    printf("x = %s, y = %s\n", x, y);  
    return 0;  
}
```

| | Address | Value |
|---|---------|-------|
| x | 0xff18 | ... |
| | 0xe | 0xe |
| y | 0xff10 | 0xc |
| | 0xf | ... |
| | 0xe | '\0' |
| | 0xd | '5' |
| | 0xd | '\0' |
| | 0xc | '2' |
| | ... | ... |

main()

DATA SEGMENT



**"Awesome! Thanks. We
also have 20 custom struct
types. Could you write
swap for those too?"**



Generic Swap

What if we could write *one* function to swap two values of any single type?

```
void swap_int(int *a, int *b) { ... }
void swap_float(float *a, float *b) { ... }
void swap_size_t(size_t *a, size_t *b) { ... }
void swap_double(double *a, double *b) { ... }
void swap_string(char **a, char **b) { ... }
void swap_mystruct(mystruct *a, mystruct *b) { ... }

...
```

Generic Swap

```
void swap_int(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
void swap_short(short *a, short *b) {  
    short temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
void swap_string(char **a, char **b) {  
    char *temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

Generic Swap

```
void swap_int(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void swap_short(short *a, short *b) {
    short temp = *a;
    *a = *b;
    *b = temp;
}

void swap_string(char **a, char **b) {
    char *temp = *a;
    *a = *b;
    *b = temp;
}
```

All three of these:

- Take pointers to values to that should be exchanged
- Create temporary storage to store one of the values
- Move data addressed by **b** into the space addressed by **a**
- Move copy of temporary into spaces addressed by **b**

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

```
int temp = *data1ptr;
```

4 bytes

```
short temp = *data1ptr;
```

2 bytes

```
char *temp = *data1ptr;
```

8 bytes

Problem: each type may need a different size temp!

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

*data1Ptr = *data2ptr; 4 bytes

*data1Ptr = *data2ptr; 2 bytes

*data1Ptr = *data2ptr; 8 bytes

Problem: each type needs to copy a different amount of data!

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

*data2ptr = temp;

4 bytes

*data2ptr = temp;

2 bytes

*data2ptr = temp;

8 bytes

Problem: each type needs to copy a different amount of data!

**C knows the size of `temp`,
and knows how many bytes
to replicate, because of the
variable types.**



**Is there a way to make a
version that doesn't care
about the variable types?**

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(pointer to data1, pointer to data2) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr) {  
    store a copy of data1 in temporary storage  
    copy data2 to location of data1  
    copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr) {  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr) {  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

If we don't know the data type, we don't know how many bytes it is. Let's take that as another parameter.

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

If we don't know the data type, we don't know how many bytes it is. Let's take that as another parameter.

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Let's start by making space to store the temporary value. How can we make **nbytes** of temp space?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void temp; ???  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Let's start by making space to store the temporary value. How can we make **nbytes** of temp space?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

temp is **nbytes** of memory,
since each **char** is 1 byte!

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Now, how can we copy in what
data1ptr points to into **temp**?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    temp = *data1ptr; ???  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Now, how can we copy in what
data1ptr points to into **temp**?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    temp = *data1ptr; ???  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

We can't dereference a **void *** (or set an array equal to something). C doesn't know what it points to! Therefore, it doesn't know how many bytes there it should be looking at.

memcpy

memcpy is a function that copies a specified amount of bytes at one address to another address.

```
void *memcpy(void *dest, const void *src, size_t n);
```

It copies the next n bytes that src points to to the location contained in dest. (It also returns **dest**). It does not support regions of memory that overlap.

```
int x = 5;  
int y = 4;  
memcpy(&x, &y, sizeof(x)); // like x = y
```

memcpy must take **pointers** to the bytes to work with to know where they live and where they should be copied to.

memmove

memmove is the same as **memcpy**, except it handles overlapping memory figures.

```
void *memmove(void *dest, const void *src, size_t n);
```

It copies the next n bytes that src points to to the location contained in dest. (It also returns **dest**).

memmove

When might **memmove** be useful?

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|



| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 5 | 6 | 7 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    temp = *data1ptr; ???  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

We can't dereference a **void ***. C doesn't know what it points to! Therefore, it doesn't know how many bytes there it should be looking at.

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    temp = *data1ptr; ???  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

How can **memcpy** or **memmove** help us here? (Assume data to be swapped is not overlapping).

```
void *memcpy(void *dest, const void *src, size_t n);
```

```
void *memmove(void *dest, const void *src, size_t n);
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

We can copy the bytes ourselves into temp! This is equivalent to **temp = *data1ptr** in non-generic versions, but this works for *any* type of *any* size.

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    // copy data in temporary storage to location of data2  
}
```

How can we copy data2 to the location of data1?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    *data1ptr = *data2ptr; ???  
    // copy data in temporary storage to location of data2  
}
```

How can we copy data2 to the location of data1?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
}
```

How can we copy data2 to the location of data1?
memcpy!

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
}
```

How can we copy temp's data to the location of data2?

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
    memcpy(data2ptr, temp, nbytes);  
}
```

How can we copy temp's data to the location of data2? **memcpy!**

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
    memcpy(data2ptr, temp, nbytes);  
}
```

```
int x = 2;  
int y = 5;  
swap(&x, &y, sizeof(x));
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
    memcpy(data2ptr, temp, nbytes);  
}
```

```
short x = 2;  
short y = 5;  
swap(&x, &y, sizeof(x));
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
    memcpy(data2ptr, temp, nbytes);  
}
```

```
char *x = "2";  
char *y = "5";  
swap(&x, &y, sizeof(x));
```

Generic Swap

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    // store a copy of data1 in temporary storage  
    memcpy(temp, data1ptr, nbytes);  
    // copy data2 to location of data1  
    memcpy(data1ptr, data2ptr, nbytes);  
    // copy data in temporary storage to location of data2  
    memcpy(data2ptr, temp, nbytes);  
}
```

```
mystruct x = {...};  
mystruct y = {...};  
swap(&x, &y, sizeof(x));
```

C Generics

- We can use **void *** and **memcpy** to manipulate raw memory.
- If we know where the data is and how big it is, we can manipulate it!

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    char temp[nbytes];  
    memcpy(temp, data1ptr, nbytes);  
    memcpy(data1ptr, data2ptr, nbytes);  
    memcpy(data2ptr, temp, nbytes);  
}
```

void *, memcpy, memmove

From a design standpoint, why does **memcpy** take **void ***s as parameters?

```
int x = 2;  
int y = 3;  
memcpy(&x, &y, sizeof(x)); // copy 3 into x
```

```
// why not this?  
memcpy(x, y);
```

1. The first parameter must be a pointer so **memcpy** knows where to copy to.
2. The second parameter *could* be a non-pointer. But then there must be a version of **memcpy** for every possible type we would like to copy!

memcpy_i(void *, int); **memcpy_c(void *, char);** **memcpy_d(void *, double);**

Void * Pitfalls

- **void ***s are powerful, but dangerous - C can't do any type checking!
- With **ints**, for example, C would never let you swap *half* of an int. With **void ***s, this can happen! (*How? Let's find out!*)

Demo: Void *s Gone Wrong



swap.c

Void * Pitfalls

- void * has more room for error because it manipulates arbitrary bytes without knowing what they represent. This can result in some strange memory Frankensteins!



<http://i.ytimg.com/vi/10gPoYjq3EA/hqdefault.jpg>