

COMPUTER SYSTEMS AND ORGANIZATION

Generic Swap, Memcmp

Daniel G. Graham Ph.D



UNIVERSITY
of VIRGINIA

ENGINEERING



1. Being careful with C functions fscanf, example
2. Generic Swap
3. Memcp, and other mem operations
4. Memory Error Puzzle strsep

A SIMPLE PRINT EXAMPLE

```
int printf(const char *format, ...);
```

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char cs_array[] = "C.S.O is fun";
    printf("%s", cs_array);
    printf("\n");
}
```

What gets printed?

YEAH THAT'S RIGHT

<pre>ssh portal.cs.virginia.edu GNU nano 6.3 simpleprint.c #include <stdio.h> #include <stdlib.h> int main(){ char cs_array[] = "C.S.O is fun"; printf("%s", cs_array); printf("\n"); return EXIT_SUCCESS; }</pre>	<pre>??M? -zsh dgg6b@portal08:~/Lecture-Code/lecture-34\$ clang simpl dgg6b@portal08:~/Lecture-Code/lecture-34\$./a.out C.S.O is fun dgg6b@portal08:~/Lecture-Code/lecture-34\$</pre>
--	--

SOMETIMES C FUNCTIONS AREN'T INTUITIVE

Let's start by looking at fscanf

```
int fscanf(FILE *stream, const char *format, ...);
```

WHAT DO WE THINK THE FOLLOWING SNIPPET OF CODE DOES

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char line[255];
    FILE *fptr = fopen("haiku.txt", "r");
    fscanf(fptr, "%s", line); // What does line contain?
    printf("%s", line);      // What this line print
    print("\n");
}
```

haiku.txt

C.S.O. is fun
Knowledge blooms as the end looms
It is almost done

```
~ — ssh portal.cs.virginia.edu
GNU nano 6.3      fprintfExample.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char line[255];
    FILE *fptr = fopen("haiku.txt", "r");
    fscanf(fptr, "%s", line); // What does line contain?
    printf("%s", line); // What this line print
    printf("\n");
}
```

```
~ — ??M? — -zsh
dgg6b@portal08:~/Lecture-Code/lecture-34$ clang fprintfExample.c
dgg6b@portal08:~/Lecture-Code/lecture-34$ ./a.out
C.S.O.
dgg6b@portal08:~/Lecture-Code/lecture-34$
```

Wait what just prints C.S.O
Why....?

haiku.txt

C.S.O. is fun
Knowledge blooms as the end looms
It is almost done

LET'S CHECK THE DOCUMENTATION

X	Equivalent to x .	
f	Matches an optionally signed floating-point number; the next pointer must be a pointer to <u>float</u> .	
e	Equivalent to f .	
g	Equivalent to f .	Different behavior for fscanf And printf
E	Equivalent to f .	
a	(C99) Equivalent to f .	
s	Matches a sequence of non-white-space characters; the next pointer must be a pointer to the initial element of a character array that is long enough to hold the input sequence and the terminating null byte ('\0'), which is added automatically. The input string stops at white space or at the maximum field width, whichever occurs first.	

GENERAL GUIDANCE ON CHOOSING FUNCTIONS

Read the documentation closely 😊

That's it. C was first so let's learn to love all its quirks and features and then program in RUST 😊

ANOTHER FUN EXAMPLE

```
char *strsep(char **stringp, const char *delim);
```

Stringp is a pointer
to string that we
want to

delim is a string that
contains multiple
delimiters

SWAP

```
#include <stdio.h>
```

```
void swapShorts(short *x, short *y) {  
    short temp = *x;  
    *x = *y;  
    *y = temp;  
}
```

```
int main() {  
    short a = 10, b = 20;  
    printf("Before swapping: a = %d, b = %d\n", a, b);  
  
    swapShorts(&a, &b);  
  
    printf("After swapping: a = %d, b = %d\n", a, b);  
  
    return 0;  
}
```

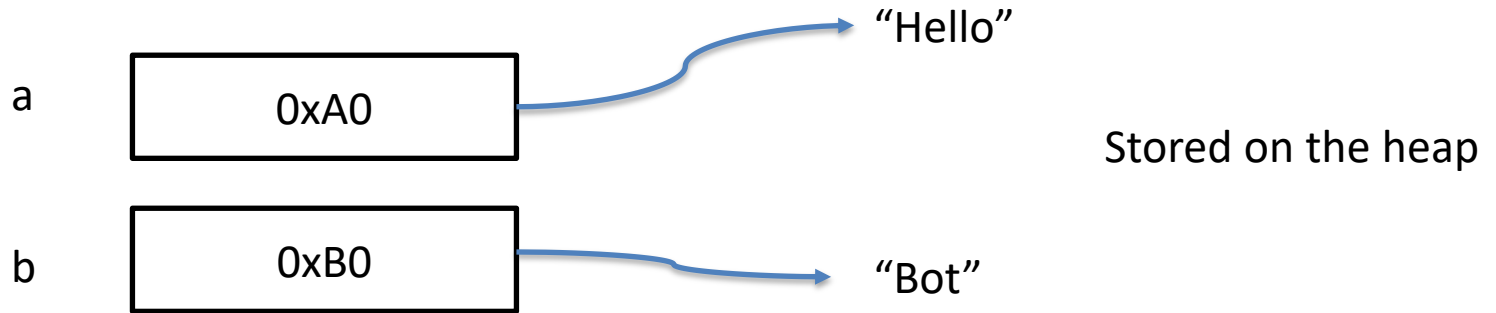
SWAP TO STRINGS

How could you do this?

SWAP TO STRINGS

How could you do this?
Just swap the value of the pointers ;)

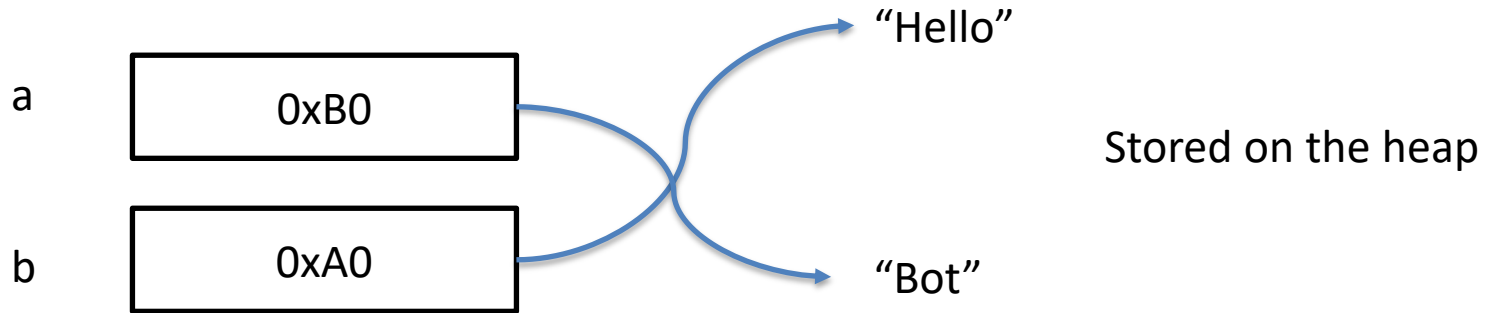
```
char* a = strdup("Hello");  
char* b = strdup("Bot");
```



SWAP TO STRINGS

How could you do this?
Just swap the value of the pointers ;)

```
char* a = strdup("Hello");  
char* b = strdup("Bot");
```



SWAP STRINGS

```
#include <stdio.h>

void swapStrings(char **str1, char **str2) {
    char *temp = *str1;
    *str1 = *str2;
    *str2 = temp;
}

int main() {
    char *a = "Hello";
    char *b = "Bot";

    printf("Before swapping: a = %s, b = %s\n", a, b);
    swapStrings(&a, &b);
    printf("After swapping: a = %s, b = %s\n", a, b);

    return 0;
}
```

GENERIC SWAP

Could we write a generic that could pass any two pointers and then have the memory pointer swap

MEMCPY

memcpy is a function in the C standard library, defined in the header file <string.h>. It is used to copy a specified number of bytes from one memory location to another.

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

- dest: Pointer to the destination array where the content is to be copied.
- src: Pointer to the source of data to be copied.
- n: Number of bytes to copy.
- Returns a pointer to dest.

MEMCPY EXAMPLE

```
#include <stdio.h>
#include <string.h>

int main() {
    char src[50] = "This is the source string.";
    char dest[50];

    // Copy src to dest
    // Added to make we get the null terminator
    memcpy(dest, src, strlen(src) + 1);
    printf("dest = \"%s\"\n", dest);

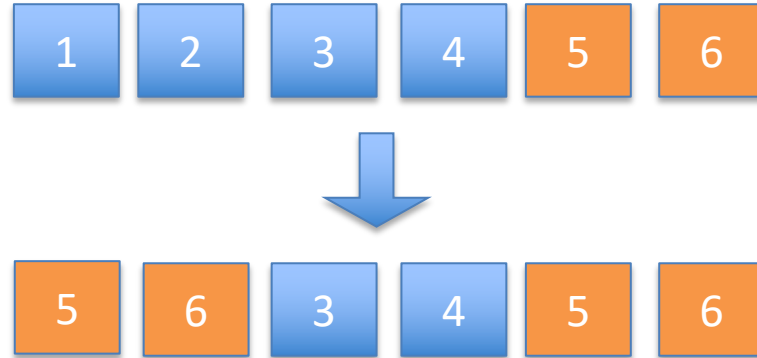
    return 0;
}
```

MEMMOVE

It is similar to memcpy, but the key difference is that memmove is safe to use when the source and destination memory regions overlap. This is because memmove takes care of the possibility of overlapping regions by ensuring that the copying of bytes does not interfere with the original content in the case of overlap.

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

MEMMOVE



TALK TO YOUR NEIGHBOR

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[50] = "Hello, World!";

    printf("Original string: %s\n", str);
    memmove(str + 7, str + 0, 5);

    printf("After memmove: %s\n", str);

    return 0;
}
```

What get's printed after memmove?

TALK TO YOUR NEIGHBOR

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[50] = "Hello, World!";

    printf("Original string: %s\n", str);
    memmove(str + 7, str + 0, 5);

    printf("After memmove: %s\n", str);

    return 0;
}
```

What get's printed after memmove?

Answer: Hello, Hello!

BACK TO GENERIC SWAP

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

BACK TO GENERIC SWAP

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


BACK TO GENERIC SWAP

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

BACK TO GENERIC SWAP

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

BACK TO GENERIC SWAP

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

BACK TO GENERIC SWAP

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

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void *temp = malloc(nbytes);  
    memcpy(temp, data1ptr, nbytes);  
    copy data2 to the location of data1  
    copy data in temporary storage to the location of data2  
}
```

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void *temp = malloc(nbytes);  
    memcpy(temp, data1ptr, nbytes);  
    memcpy(data1ptr, data2ptr, nbytes);  
    copy data in temporary storage to the location of data2  
}
```

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void *temp = malloc(nbytes);  
    memcpy(temp, data1ptr, nbytes);  
    memcpy(data1ptr, data2ptr, nbytes);  
    copy data in temporary storage to the location of data2  
}
```

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void *temp = malloc(nbytes);  
    memcpy(temp, data1ptr, nbytes);  
    memcpy(data1ptr, data2ptr, nbytes);  
    memcpy(data2ptr, temp, nbytes);  
}
```

Just one more thing.

BACK TO GENERIC SWAP

```
void swap(void *data1ptr, void *data2ptr, size_t nbytes) {  
    void *temp = malloc(nbytes);  
    memcpy(temp, data1ptr, nbytes);  
    memcpy(data1ptr, data2ptr, nbytes);  
    memcpy(data2ptr, temp, nbytes);  
    free(temp);  
}
```

PUZZLE 1


```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(){
    int *x = (int *) malloc(40*sizeof(int));
    x+= 5;
    free(x);
}
```

PUZZLE 1

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
int main(){
    int *x = (int *) malloc(40*sizeof(int));
    x+= 5;
    free(x);
}
```



Not the address associated with the initial malloc.

STRSEP

```
char *strsep(char **stringp, const char *delim);
```

DESCRIPTION

If *stringp is NULL, the `strsep()` function returns NULL and does nothing else. Otherwise, this function finds the first token in the string *stringp, that is delimited by one of the bytes in the string delim. This token is terminated by overwriting the delimiter with a null byte ('\0'), and *stringp is updated to point past the token. In case no delimiter was found, the token is taken to be the entire string *stringp, and *stringp is made NULL.

STRSEP

Draw visual.

```
#include <stdio.h>
#include <string.h>

int main() {
    char string[] = "a,b,c,d"; // The string to be tokenized
    char *token;
    char *rest = string;

    while ((token = strsep(&rest, ",")) != NULL) {
        printf("%s\n", token);
    }

    return 0;
}
```

PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d"); // stored on the heap
    char *token;
    token = strtok(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d"); // stored on the heap
    char *token;
    token = strtok(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

PUZZLE 2 DOES THIS CRASH OR NOT?

```
~ — ssh portal.cs.virginia.edu
GNU nano 6.3 strsepfree.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d"); // stored on the heap
    char *token;
    token = strsep(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}

~ — ???M? — -zsh
dgg6b@portal08:~/Lecture-Code/lecture-34$ clang -O3 strsepfree.c
dgg6b@portal08:~/Lecture-Code/lecture-34$ ./a.out
a
free(): invalid pointer
Aborted (core dumped)
dgg6b@portal08:~/Lecture-Code/lecture-34$
```


PUZZLE 2 DOES THIS CRASH OR NOT?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d"); // stored on the heap
    char *token;
    token = strsep(&string, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

Crashed because the string pointer is updated to a new address. It is not the original malloced address.

PUZZLE 2 (WHY DOES THIS

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char *string = strdup("a,b,c,d"); // stored on the heap
    char *token;
    char *rest = string;
    token = strtok(&rest, ",");
    printf("%s\n", token);
    free(string);
    return 0;
}
```

REFERENCES AND CREDIT

<https://web.stanford.edu/class/archive/cs/cs107/cs107.1242/lectures/12/Lecture12.pdf>

