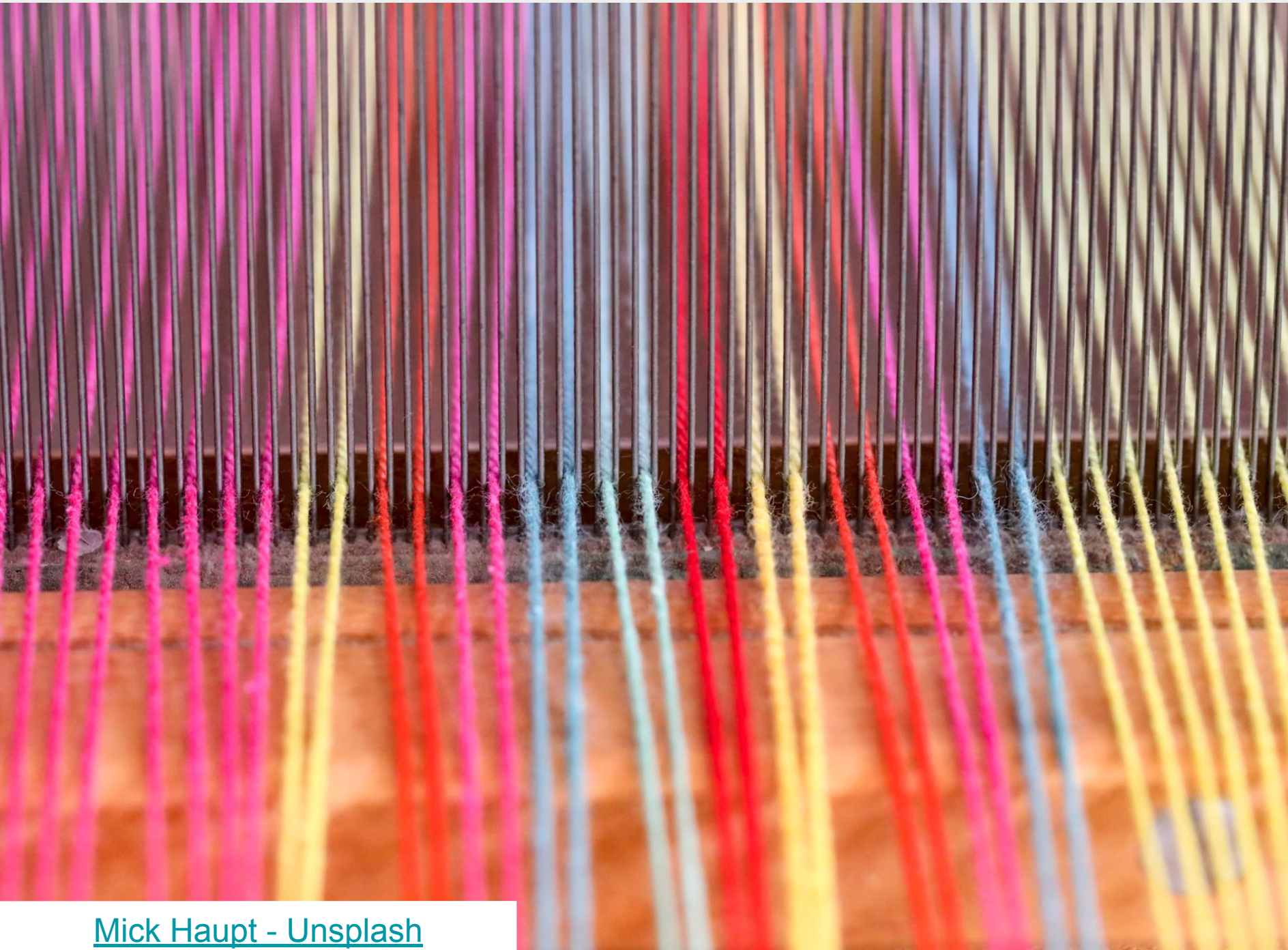
Memory and String Function in C

Adapted from materials by Dr. Carrier



Mick Haupt - Unsplash

New include!

All functions are in <string.h>

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

Memory Functions

How can we zero-out memory?

```
int* data = (int*)malloc(sizeof(int) * 32);
```

Memory Functions

How can we zero-out memory?

```
int* data = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   data[i] = 0
}
```

Memory Functions

How can we zero-out memory?

```
int* data = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   data[i] = 0
}
```

Alternative:

`memset(void* s, int c, size_t n);`

Fills *n* **bytes** of *s* with **byte** *c*

Memory Functions

How can we zero-out memory?

```
int* data = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   data[i] = 0
}
```

Alternative:

`memset(void* s, int c, size_t n);`

Fills *n* **bytes** of *s* with **byte** *c*

```
memset(data, 0, 32 * sizeof(int));
```

Memory Functions

How can we zero-out memory?

```
int* data = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   data[i] = 0
}
```

Alternative:

`memset(void* s, int c, size_t n);`

Fills *n* **bytes** of *s* with **byte** *c*

```
memset(data, 0, 32 * sizeof(int));
```

Does this work?

```
memset(data, 1, 32 * sizeof(int));
```

Memory Functions

How can we copy an array?

```
int* data = (int*)malloc(sizeof(int) * 32);
int* copy = (int*)malloc(sizeof(int) * 32);
```

Memory Functions

How can we copy an array?

```
int* data = (int*)malloc(sizeof(int) * 32);
int* copy = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   copy[i] = data[i];
}
```

Memory Functions

How can we copy an array?

```
int* data = (int*)malloc(sizeof(int) * 32);
int* copy = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   copy[i] = data[i];
}
```

Alternative:

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

Copy *n* **bytes**

Memory Functions

How can we copy an array?

```
int* data = (int*)malloc(sizeof(int) * 32);
int* copy = (int*)malloc(sizeof(int) * 32);
for(int i = 0; i < 32; i++){
   copy[i] = data[i];
}
```

Alternative:

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

Copy *n* **bytes**

```
memcpy(copy, data, 32 * sizeof(int));
```

Memory Functions

How can we copy an array?

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

Copy *n* **bytes**

Another alternative:

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

Copy *n* **bytes**, allow for overlapping buffers

```
memmove(copy, data, 32 * sizeof(int));
```

String functions

How are strings different than other arrays?

String functions

How are strings different than other arrays?

- Null terminator! \0

String functions

How are strings different than other arrays?

Null terminator! \0

What string functions would we like to have?

String functions - Concatenation

```
strcat(char *s1, char* s2);
```

Appends copy of s2 to s1

String functions - Concatenation

```
strcat(char *s1, char* s2);
```

Appends copy of s2 to s1

What happens if s1 isn’t big enough?

String functions - Concatenation

```
strcat(char *s1, char* s2);
```

Appends copy of s2 to s1

What happens if s1 isn't big enough?

We are writing into arbitrary memory…

String functions - Concatenation

```
strcat(char *s1, char* s2);
```

Appends copy of s2 to s1

What happens if s1 isn't big enough?
We are writing into arbitrary memory…

```
strncat(char* s1, char* s2, size_t n);
```

Same, but will only copy n chars + terminator

String functions - Concatenation

Can also use string formatting

String functions - Concatenation

Can also use string formatting

```
sprintf(char *s1, char* format_str, ...);
```

Same idea and formatting as `printf`

String functions - Concatenation

Can also use string formatting

```
sprintf(char *s1, char* format_str, ...);
```

Same idea and formatting as `printf`

Stores result in s1, adds terminator

Returns new length of s1

String functions - Concatenation

Can also use string formatting

`sprintf(char *s1, char* format_str, ...);`

Same idea and formatting as `printf`

Stores result in s1, adds terminator

Returns new length of s1

```
sprintf(s, “x = %d”, 5);
```

String functions - Copying

String functions - Copying

```
strcpy(char* dest, char* src);
```

Copies source to dest, including ‘\0’, too.

Make sure dest has enough space!

String functions - Copying

```
strcpy(char* dest, char* src);
```

Copies source to dest, including ‘\0’, too.

Make sure dest has enough space!

```
strncpy(char* dest, char* src, size_t n);
```

Copy up to n chars

String functions - Copying

```
strcpy(char* dest, char* src);
```

Copies source to dest, including ‘\0’, too.

Make sure dest has enough space!

```
strncpy(char* dest, char* src, size_t n);
```

Copy up to n chars

Will add ‘\0’s to fill if src is too short

Otherwise no terminators

String functions - Length

```
strlen(char* s);
```

Returns length of string

String functions - Length

```
strlen(char* s);
```

Returns length of string

How does this work?

String functions - Length

```
strlen(char* s);
```

Returns length of string

How does this work?

- Returns number of characters before terminator!
  (must be null-terminated!)

String functions - Comparison

```
int strcmp(char* s1, char* s2);
```

String functions - Comparison

```
int strcmp(char* s1, char* s2);
```

Lexicographically compares strings

Return 0 if strings are the same

String functions - Comparison

```
int strcmp(char* s1, char* s2);
```

- Lexicographically compares strings
  - Return 0 if strings are the same
  - Returns negative num if s1 < s2
  - Returns positive num if s1 > s2

String functions - Comparison

```
int strcmp(char* s1, char* s2);
```

Lexicographically compares strings

Return 0 if strings are the same

Returns negative num if s1 < s2

Returns positive num if s1 > s2

```
int strncmp(char* s1, char* s2, size_t n);
```

Same, but limits to n chars (or to \0)

String functions - Searching

```
char* strchr(char *s, int c);
```

Searches for first instance of char c in string

String functions - Searching

```
char* strchr(char *s, int c);
```

Searches for first instance of char c in string

Returns pointer to first instance

Returns NULL if c not found

String functions - Searching

```
char* strchr(char *s, int c);
```

Searches for first instance of char c in string
Returns pointer to first instance
Returns NULL if c not found

```
char* strrchr(char *s, int c);
```

Returns *last* instance of c in s (or NULL if none)

String functions - Searching

```
char* strstr(char* s1, char* s2);
```

String functions - Searching

```
char* strstr(char* s1, char* s2);
```

Searches for substring s2 in s1.

Returns pointer to first instance, or NULL if not found