	Pointer	Array
Declaration/ initialization	`int *ptr_to_x = &x;`	`int Arr1[] = {4, 2, 3};`

An array is like a pointer

- Accessing/writing content

	`y = *ptr_to_x;`	`y = Arr1[0];`
	`*ptr_to_x = 99;`	`Arr1[1] = 5;`
Unrecommended!	`ptr_to_x[0] = 99;`	`*(Arr1 + 1) = 5;`

- Passing address to a function

`scanf("%i", ptr_to_x);`	`scanf("%i", Arr1 + 1);`

An array is slightly different from a pointer

```
char *t = "How big is it?";
```

This creates an 8-byte pointer variable to store the address of the static string data

```
sizeof(t)
```
➡ 8

```
&t
```
➡ pointer of pointer

```
t[4] = 'B';
```
✗

Static data cannot be changed!

```
t = s;
```
✓

A pointer can be re-assigned

```
char s[] = "How big is it?";
```

This creates a 15-byte char array to store a copy of the string data

```
sizeof(s)
```
➡ 15

```
&s
```
➡ Pointer to the first elem. Same as s

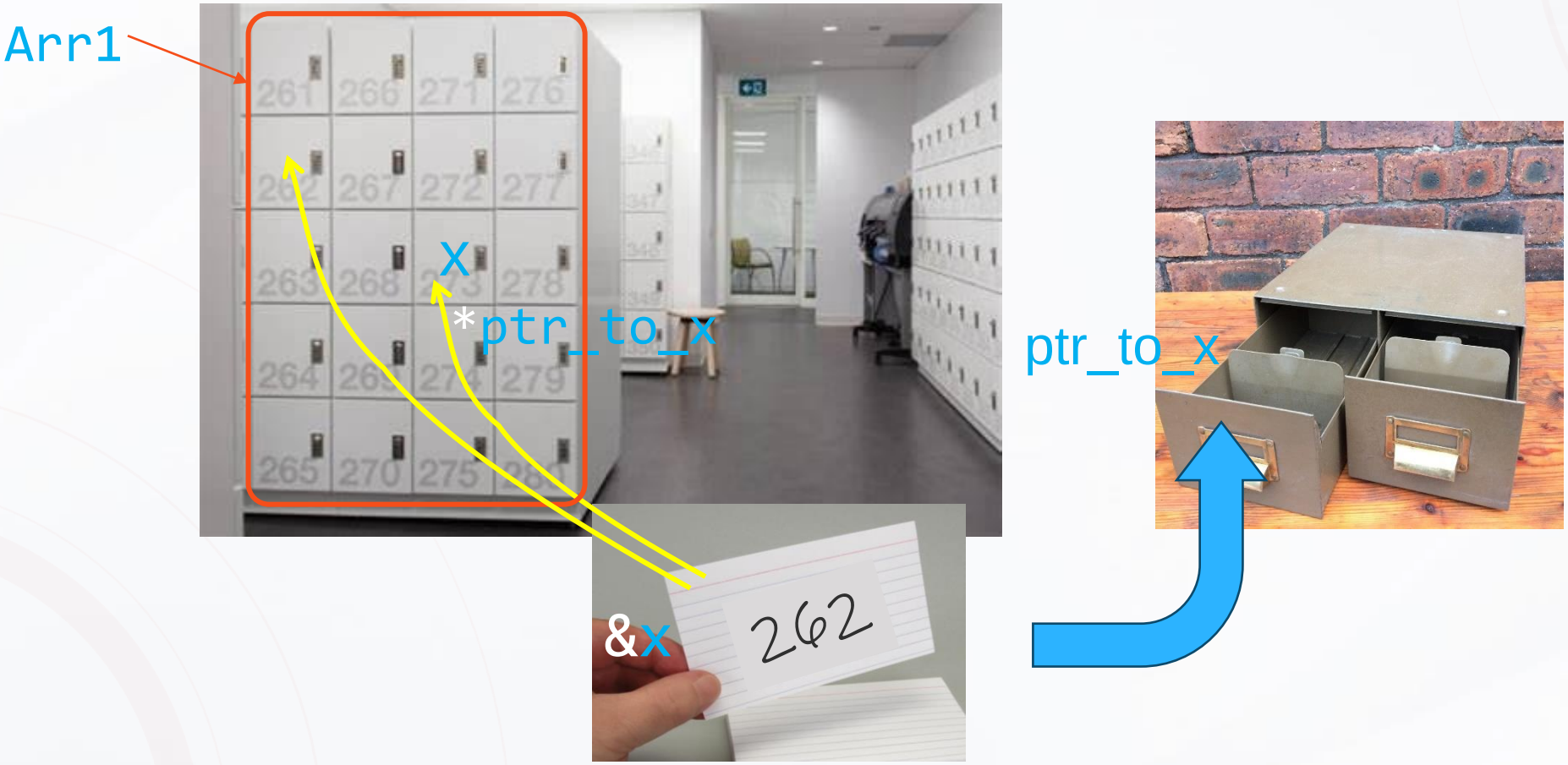
```
s[4] = 'B';
```
✓

Content of an array can be changed

```
s = t;
```
✗

Name of an array cannot be changed!

Pointer decay: an array loses it length info when passed to a function.



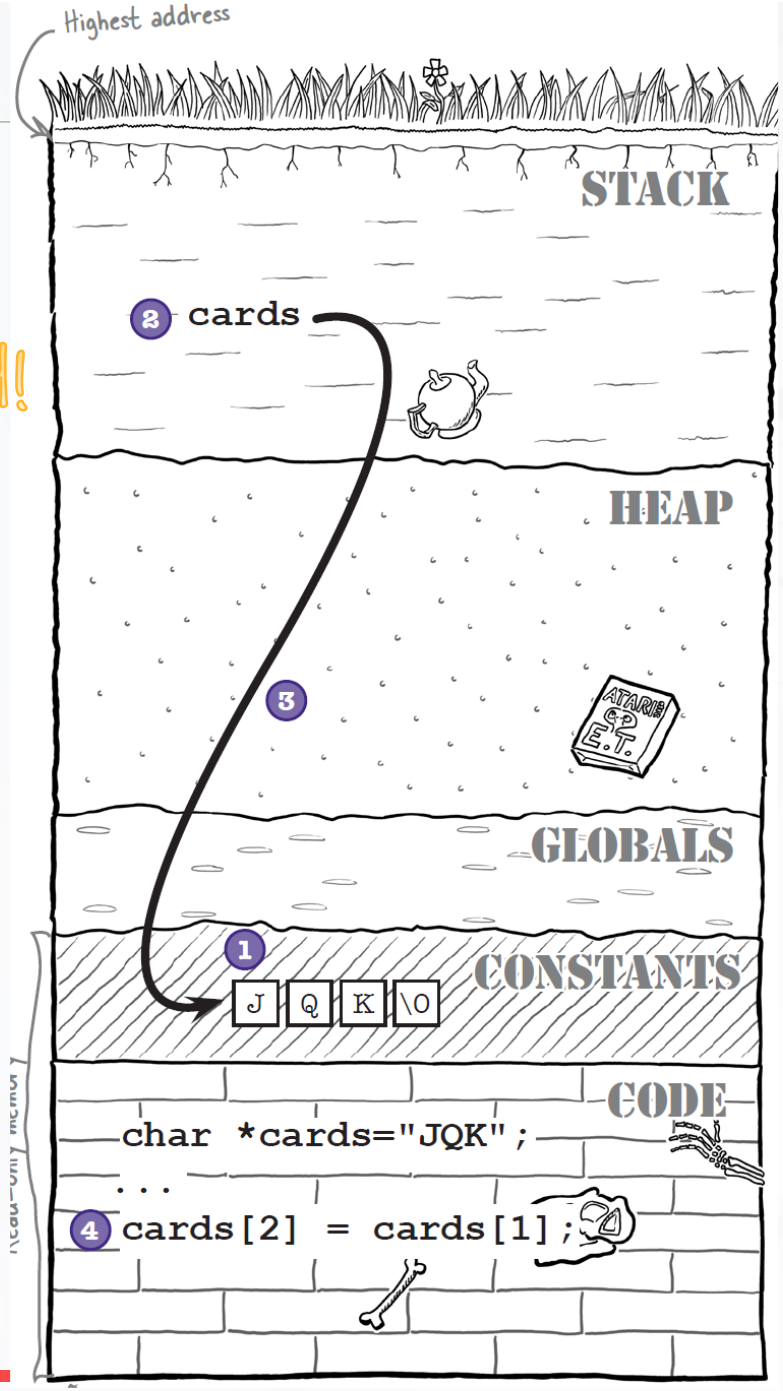
ptr_to_x = Arr1 + 1;

Three-card monte

```c
#include <stdio.h>

int main()
{
    char *cards = "JQK";
    char a_card = cards[2];
    cards[2] = cards[1];
    cards[1] = cards[0];
    cards[0] = cards[2];
    cards[2] = cards[1];
    cards[1] = a_card;
    puts(cards);
    return 0;
}
```

String literals can't be updated!



If you are going to change a string, make a copy!

```
char cards[] = "JQK";
```

Creates a char array and initialize it with the string

```
char *cards = "JQK";
```

Creates a pointer which points to the static array

Or, if you don't want to change the string, make it constant!

```
const char *cards = "JQK";
```

So, what does cards[] really mean?

It depends...

```
char cards[] = "JQK";
```

≠

```
char *cards = "JQK";
```

Declaration of an array with default length (calculated from initializer)

```
void skip(char msg[])
{
    …
}
```

=

```
void skip(char *msg)
{
    …
}
```

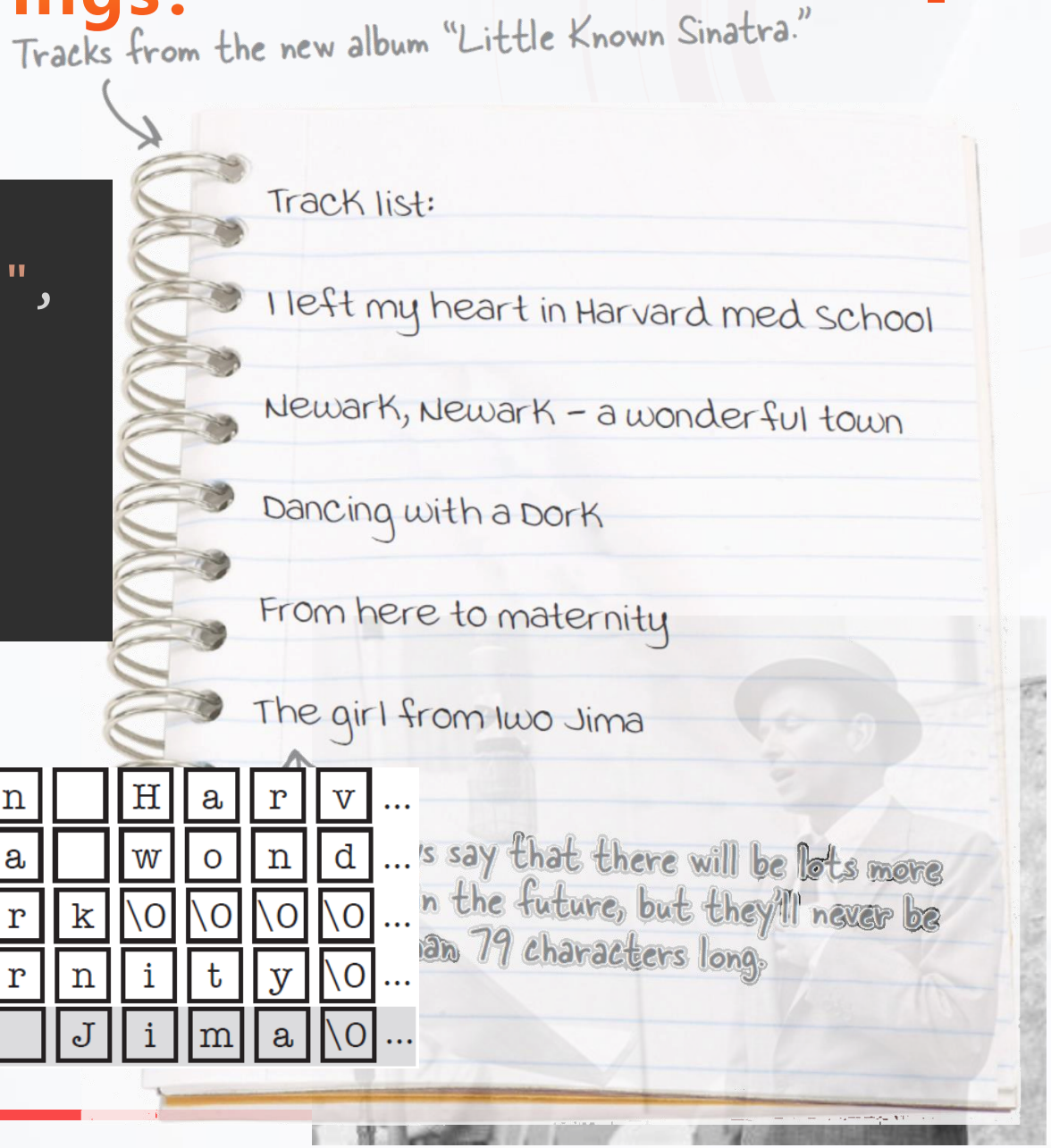
Declaration of a function argument, identical to a pointer!

Lecture 5.5 Strings

How to deal with a lot of strings?

An array of arrays (2-dimensional array)

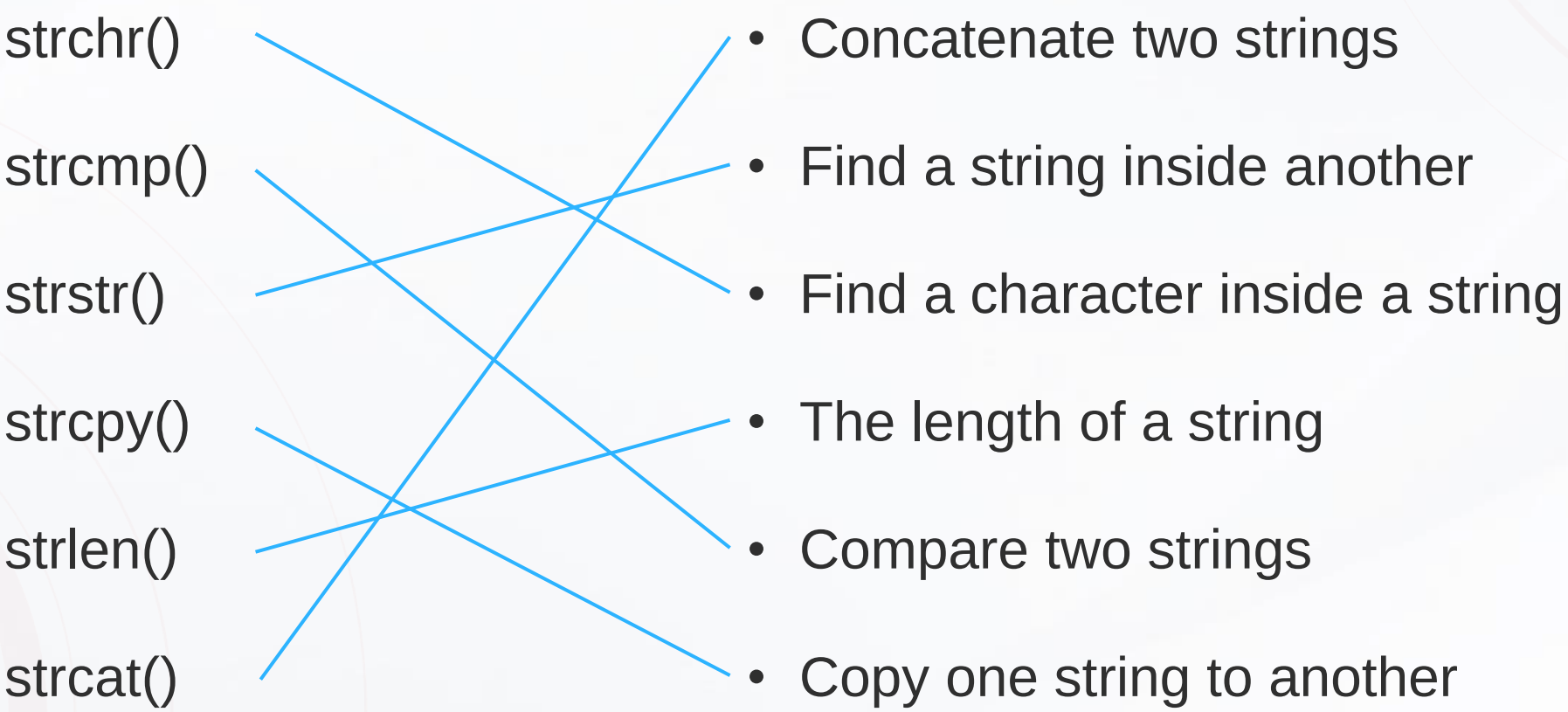
```
char tracks[][80] = {
"I left my heart in Harvard Med School",
"Newark, Newark - a wonderful town",
"Dancing with a Dork",
"From here to maternity",
"The girl from Iwo Jima",
};
```



I		l	e	f	t		m	y		h	e	a	r	t		i	n		H	a	r	v	...
N	e	w	a	r	k	,		N	e	w	a	r	k		-		a		w	o	n	d	...
D	a	n	c	i	n	g		w	i	t	h		a		D	o	r	k	\0	\0	\0	\0	...
F	r	o	m		h	e	r	e		t	o		m	a	t	e	r	n	i	t	y	\0	...
T	h	e		g	i	r	l		f	r	o	m		I	w	o		J	i	m	a	\0	...

The library functions – Guess what they mean?

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

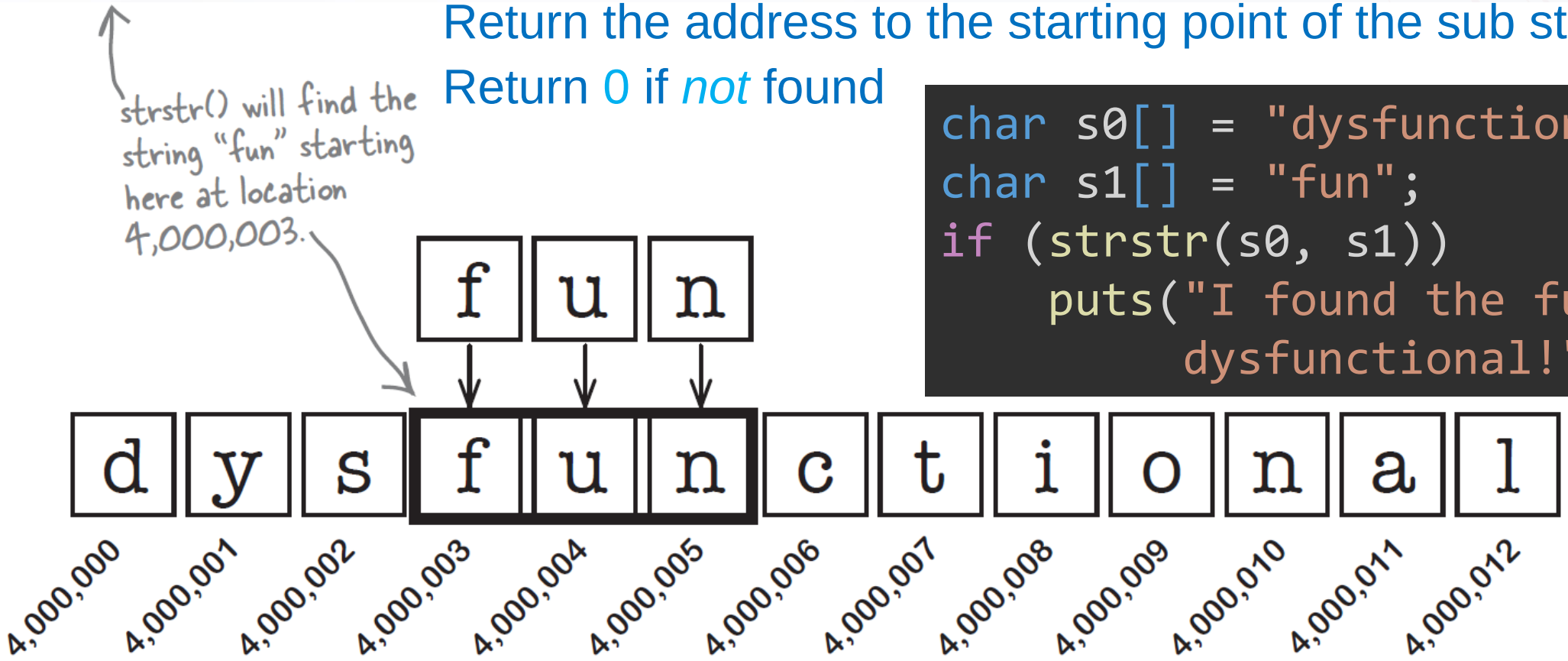


Use the strstr() function

```
strstr("dysfunctional", "fun");
```

Return the address to the starting point of the sub string

Return 0 if *not* found




```
char s0[] = "dysfunctional";
char s1[] = "fun";
if (strstr(s0, s1))
    puts("I found the fun in
          dysfunctional!");
```

Find the right track

To check whether the search term is in the track

```
void find_track(char search_for[])
{
    int i;
    for (i = 0; i < 5; i++) {
        if ( strstr( tracks[i], search_for ))
            printf("Track %i: '%s'\n",  i  , tracks[i] );
    }
}
```

Loop over all tracks

Print the track containing the search term.

Quiz – find the most appropriate program

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    fgets(search_for, 80, stdin);
    find_track();
    return 0;
}
```

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    fgets(search_for, 79, stdin);
    find_track(search_for);
    return 0;
}
```

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    fgets(search_for, 80, stdin);
    find_track(search_for);
    return 0;
}
```

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    scanf(search_for, 80, stdin);
    find_track(search_for);
    return 0;
}
```

The complete program

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

char tracks[][80] = {
    "I left my heart in Harvard Med School",
    "Newark, Newark - a wonderful town",
    "Dancing with a Dork",
    "From here to maternity",
    "The girl from Iwo Jima",
};

void find_track(char search_for[])
{
    int i;
    for (i = 0; i < 5; i++) {
        if (strstr(tracks[i], search_for))
        printf("Track %i: '%s'\n", i,
tracks[i]);
    }
}
```

Global variable

Can be accessed everywhere.

```c
int main()
{
    char search_for[80];
    printf("Search for: ");
    fgets(search_for, 80, stdin);
    find_track(search_for);
    return 0;
}
```

Array of arrays vs. array of pointers

```
char *names_for_dog[] = {"Bowser", "Bonza", "Snodgrass"};
```

This declares a 1D array that stores pointers

- Not a 2D array (no 0s are filled)!
- String literals are not to be changed!

```
char names_for_dog[][] = {"Bowser", "Bonza", "Snodgrass"};
```

This declares a 2D array that stores characters

- A 3×10 array, extra slots a filled with 0s.
- May be changed to store other strings.

Lecture 6 Functions

$$f(x) = a + bx + cx^2$$

Why functions?

- Readability – a program divided into small pieces are easier to understand and modify.
- Modulization – isolate the small pieces by allowing a few explicit connection for clarity.
- Simplicity – avoid duplicating code to be used more than once.
- Reusability – code for certain purposes can be reused in other programs.