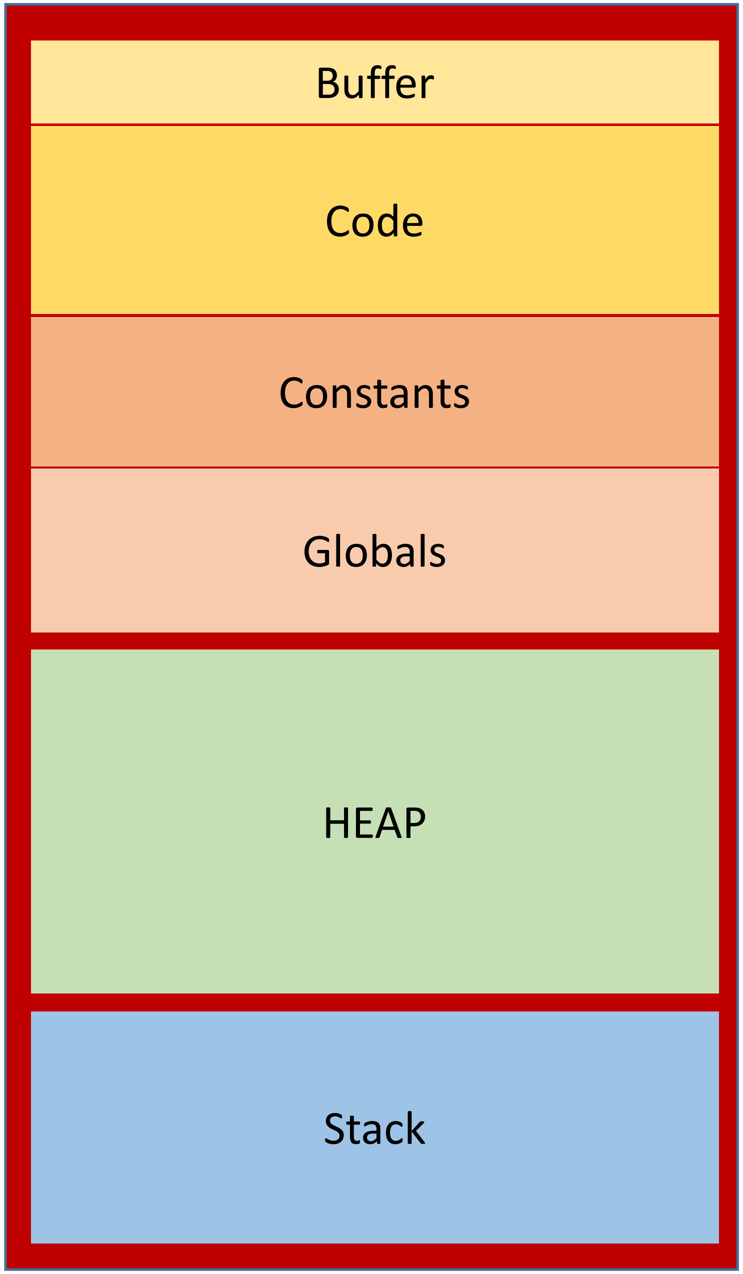
C memory model

Lecture 03.03

Outline

- Constants



Buffer

Code

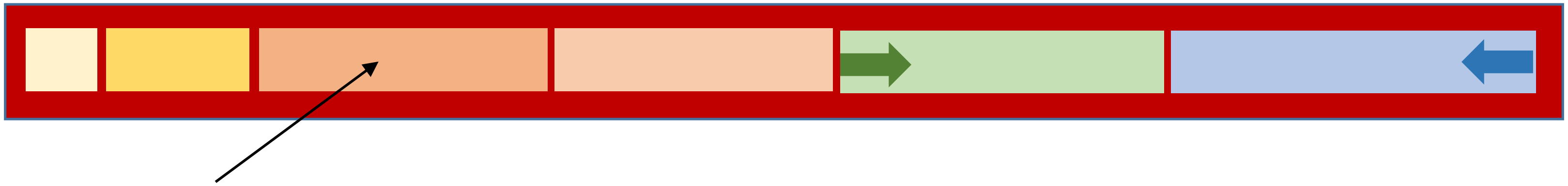
Constants

Globals

HEAP

Stack

Memory memorizer



- **Constants** stores all the constants. This memory is read-only
- **Globals** stores global variables – variables visible to all functions
- **Stack** stores variables of a currently executing function
- **Heap** is reserved for dynamic memory allocation

Three-card trick

```
#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;
}
```

Where is the Queen?

What is printed?

Compile and run: Linux

```
gcc -o trick trick.c && ./trick
bus error
```

- On different machines and operating systems:

```
trick.exe has stopped working
```

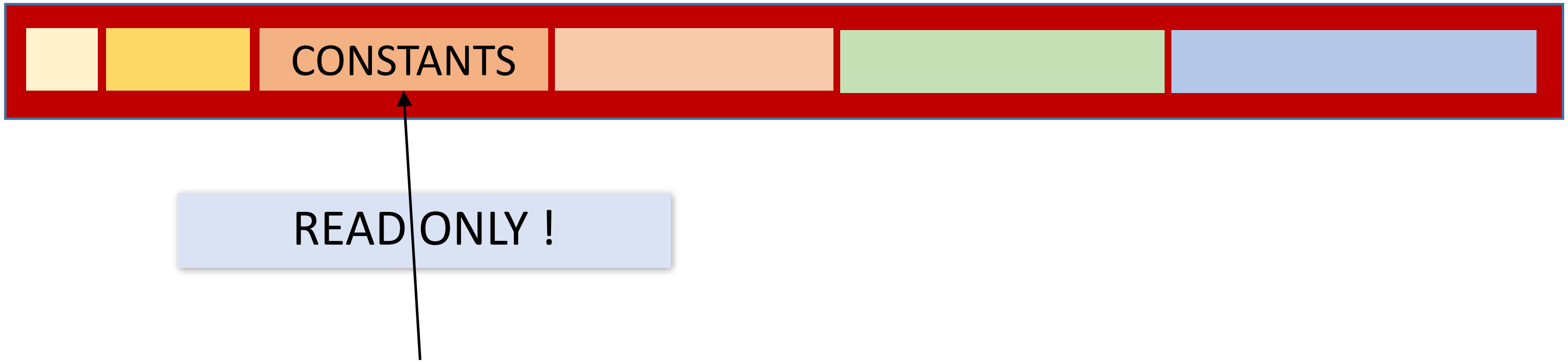
```
segmentation error
```

```
segmentation fault
```

What do you think the problem is?

A. The string can’t be updated
B. We’re swapping characters outside the string
C. The string isn’t in memory
D. Something else

String literals live in a different place: constants

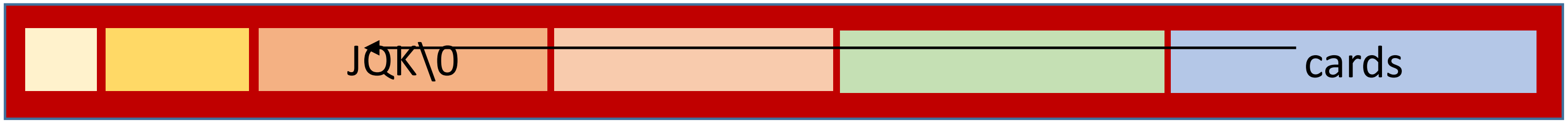

CONSTANTS

READ ONLY !

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

- We cannot update string “JQK” through pointer *cards*

String literals cannot be updated



- When the computer loads the program into memory, it puts all of the constant values—like the string literal “JQK”—into the constant memory block. This section of memory is **read only**.
- The program creates the *cards* pointer variable on the stack. The *cards* variable will contain the address of the string literal “JQK.”
- When the program tries to change the contents of the string pointed to by the *cards* variable, it can’t: the string is read-only.

Why compiler did not warn us?

- Because we declared the *cards* as a simple char *, the compiler didn’t know that the variable would always be pointing at a string literal.
- To avoid this problem never write code that sets a simple char pointer to a string literal value like:

```
char *s = "Some string";
```

✗

Correctly define pointers to string literal

`char *s = "Some string";` ✗

- There’s nothing wrong with setting a pointer to a string literal - until you try to **modify** a string literal. Instead, if you want to set a pointer to a literal, use the *const* keyword:

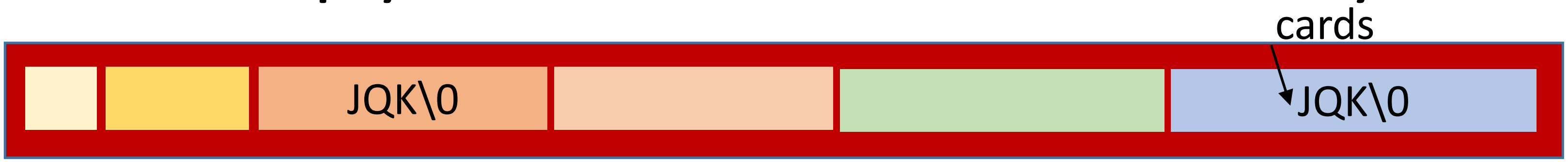
```
const char *s = "some string";
```

- That way, if the compiler sees some code that tries to modify the string, it will give you a compile error:

```
s[0] = 'S';
```

```
trick.c:7: error: assignment of read-only location
```

Fix: copy literal into char array

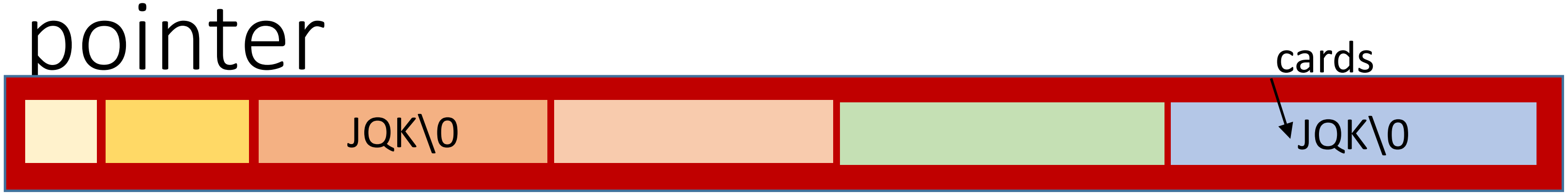


```
char cards[] = "JQK";
```

Make a copy of the string in a section of memory that can be amended

- Now cards is not a pointer. Cards is now an array, which lives on the stack. It is filled with copies of characters from the constant when the stack frame for main is loaded

If you plan to modify: use array not pointer



~~char * cards = "JDK";~~

char cards[] = "JQK";

- It’s probably not too clear why this changes anything. All strings are arrays. But in the old code, *cards* was just a pointer.
- In the new code, it’s an array. If you declare an array called *cards* and then set it to a string literal, the *cards* array will be a completely new **copy**. The variable isn’t just pointing at the string literal. It’s a brand-new array that contains a fresh copy of the string literal.

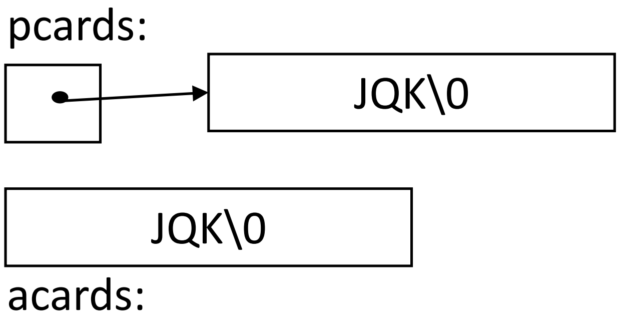
Reminder: array is not exactly a pointer

- An array name is a constant address, while a pointer is a variable:

```
int x[10], *px;
px = x; px++; /** valid **/
x = px; x++; /** invalid, cannot assign a new value **/
```

- Also, defining the pointer only allocates memory space for the address, not for any array elements, and the pointer does not point to anythingmeaningful.
- Defining an array (x[10]) gives a pointer to a specific place in memory and allocates enough space to hold the array elements.

char * vs. char []



- There is an important difference between these definitions:

char acards[] = “JQK"; /* an array */

char *pcards = “JQK"; /* a pointer */

- ***acards*** is an array, just big enough to hold the sequence of characters and ’\0’. Individual characters within the array may be changed but *acards* will always refer to the same storage.
- ***pcards*** is a pointer, initialized to point to a string constant; the pointer may subsequently be modified to point elsewhere, but the result is undefined if you try to modify the string contents.