



CIS 3990 Recitation 00

C Refresher

2025-08-28

Agenda

01 Icebreaker

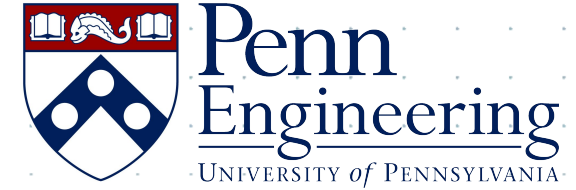
02 Logistics

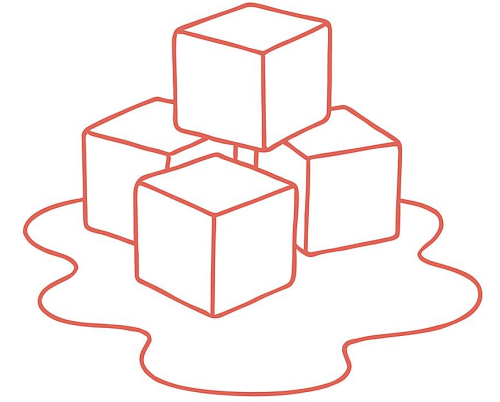
03 Pointer Review

04 Strings in C

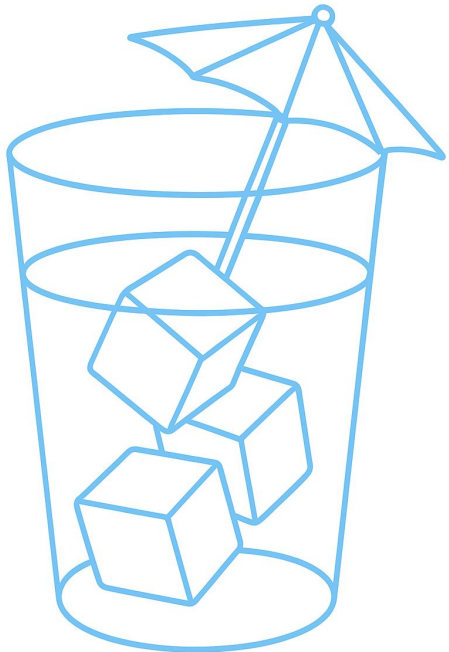
05 Ed from class

06 AMA & Cookies





Ice Breaker



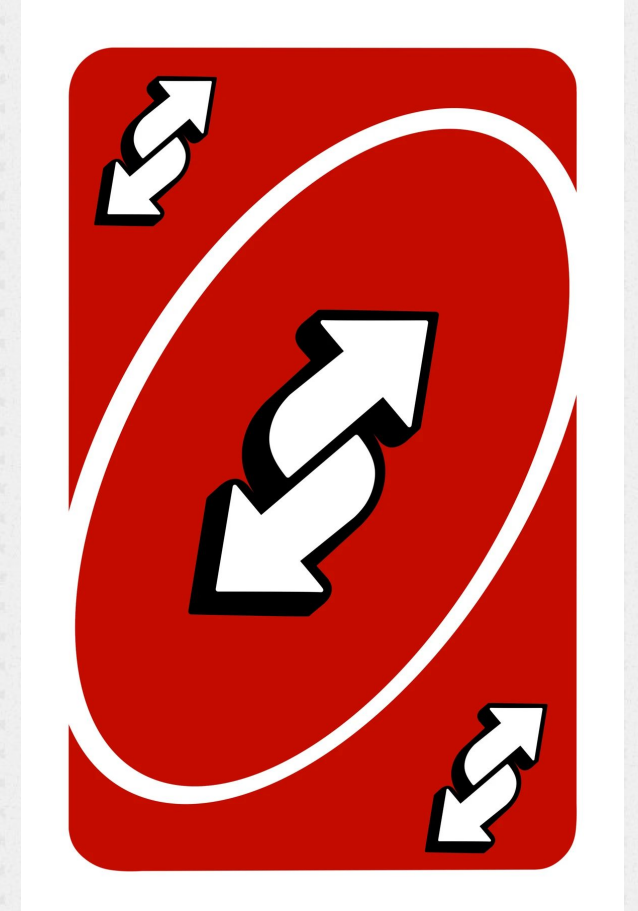
A bit about me

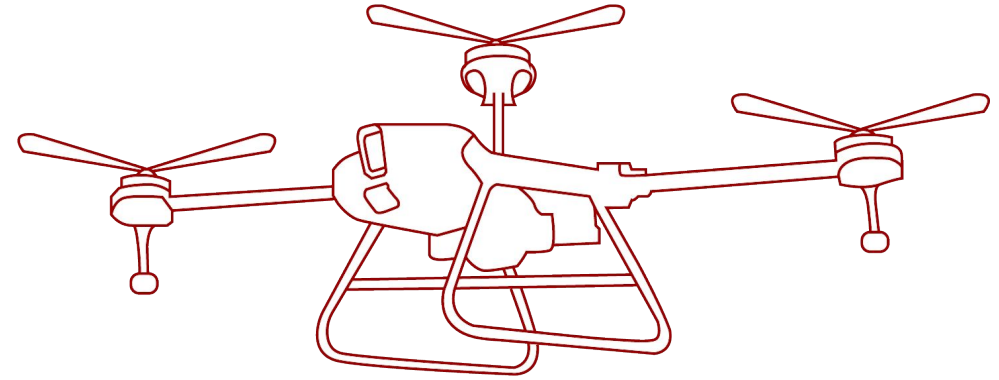
- Theodor/ Theo, he/him
- CMPE '27
- Penn Electric Racing
- Previously TA'd CIS 1100
- Tennis 🎾 & Swimming 🏊
- From Iași, Romania 🧑🏻 🇷🇴
- This summer: CompArch research
- I love cats and caffeine



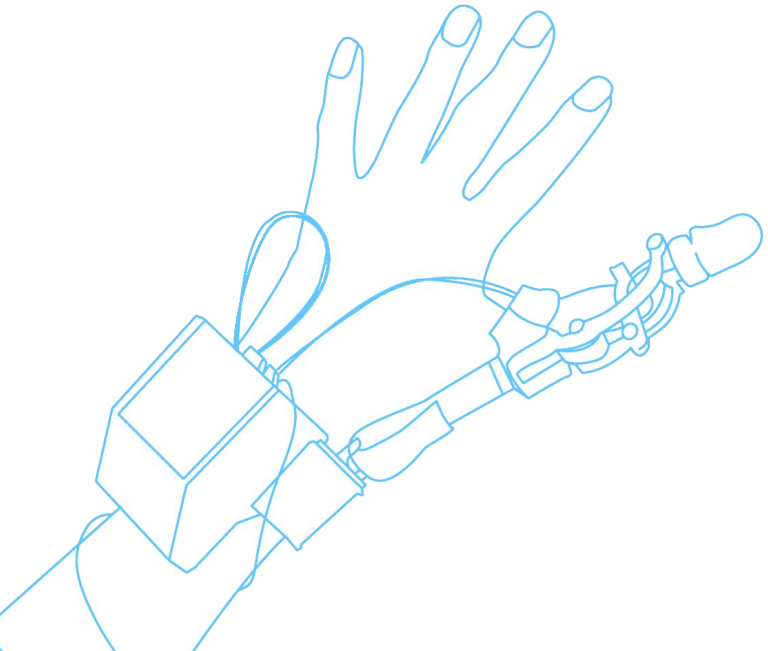
Your turn!

- Name & pronouns (if comfortable)
- Year & Major
- Hometown
- Fun fact/ hobby/ passion
- What do you want to get out of CIS 3990?





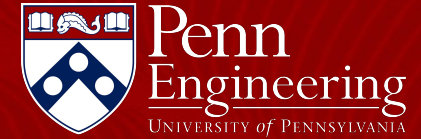
Logistics



Logistics

- HW00 out - due 11:59pm on Tues, Sep 02 - via Gradescope
- Survey00 out - due 11:59pm on Fri, Sep 12 - via Canvas
- Docker setup updated and released last night
- Github setup - automatic via Gradescope
- Oct 22 - Exam 0 - Please let us know ASAP if you have conflicts

RIGHT NOW: Gradescope Email!



Your Account

Update your account information to the right.

Logged in via LTI 1.0.

Account Settings

* Required field

Full Name *

Theodor Bulacovschi

Email Addresses

theodorb@seas.upenn.edu

★ Primary Address

[+ Add Email](#)

Have multiple accounts? [⚙ Merge Accounts](#)

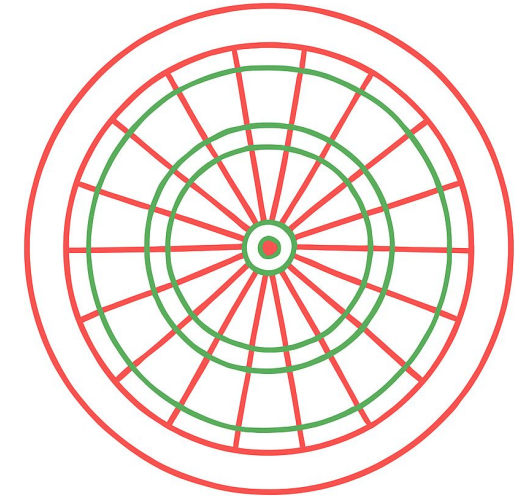
TA OH Discussion

- First TA OH: Sep 04
- Currently: Thur 2:30p - 4:30p
- Currently: Fri 4:00p-8:00p
- Location: Levine 3rd Floor “bump space” - chairs & couches by LEVH 358 - LEVH 364
- I can do less Friday and add some OH Sat/ Sun or move the Friday OH earlier in the day depending on when they are useful for **you**

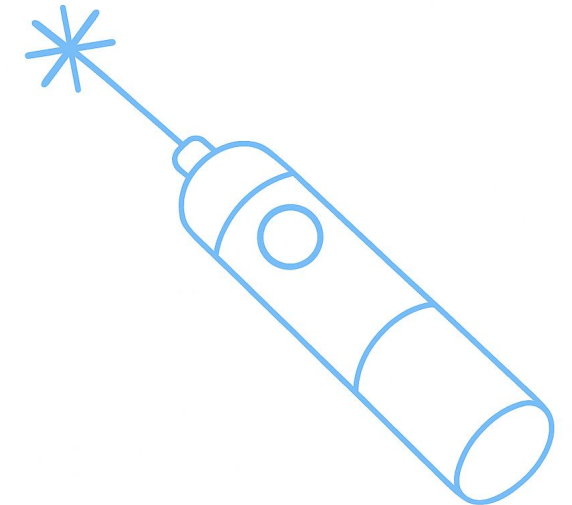
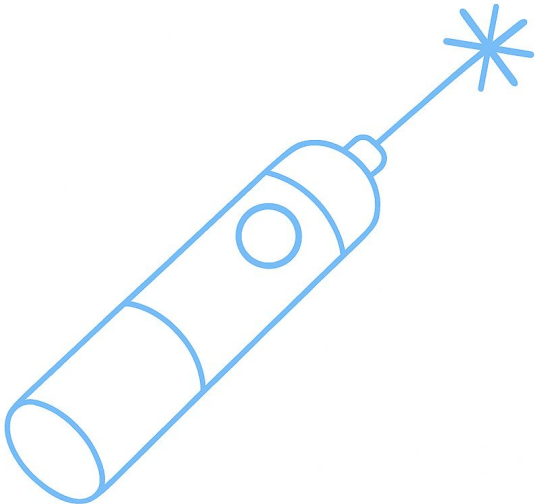
Site tour

- <https://www.cis.upenn.edu/~tqmcgaha/cis3990/25fa>





Pointer Review



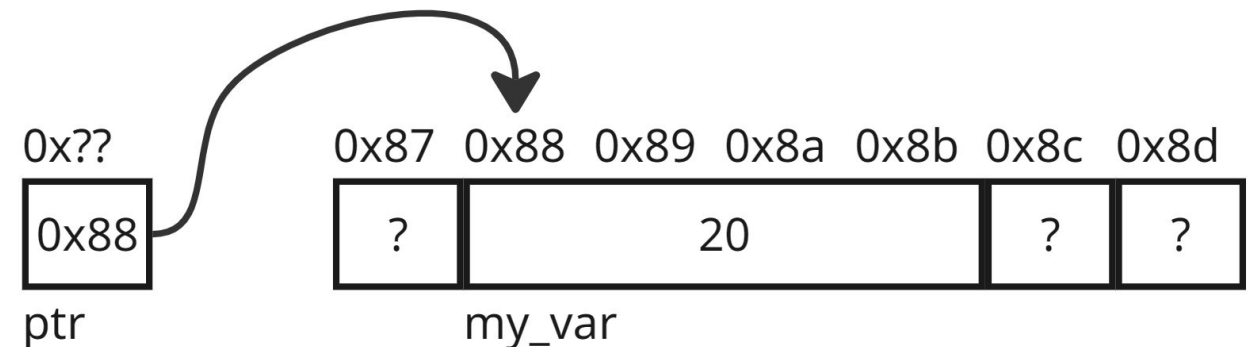
Pointers

- Primitive data type
- If you think of memory as a huge array of bytes, then a pointer is just an index
- They are also somewhere in memory

```
type *name;
```

```
int32_t my_var = 20;
```

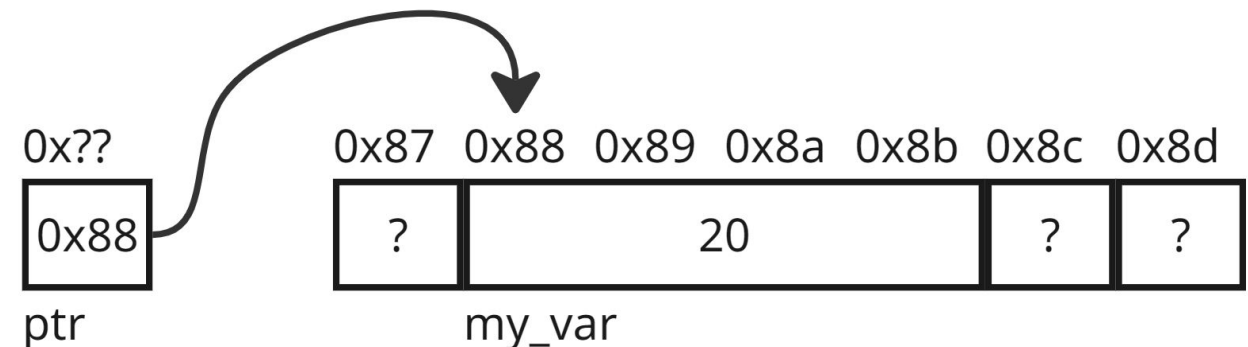
```
int32_t *ptr = &my_var;
```



Pointer Syntax

- ***** in declaration -> making it be a pointer
- ***** in usage -> accessing data at the address stored by the pointer
- **&** in usage -> getting the address of something

```
int main() {  
  
    int32_t my_var = 20;  
  
    int32_t *ptr = &my_var;  
  
    cout << my_var << endl;    // 20  
    cout << &my_var << endl; // 0x88  
    cout << ptr << endl;       // 0x88  
    cout << *ptr << endl;      // 20  
}
```



Pointer to a pointer?

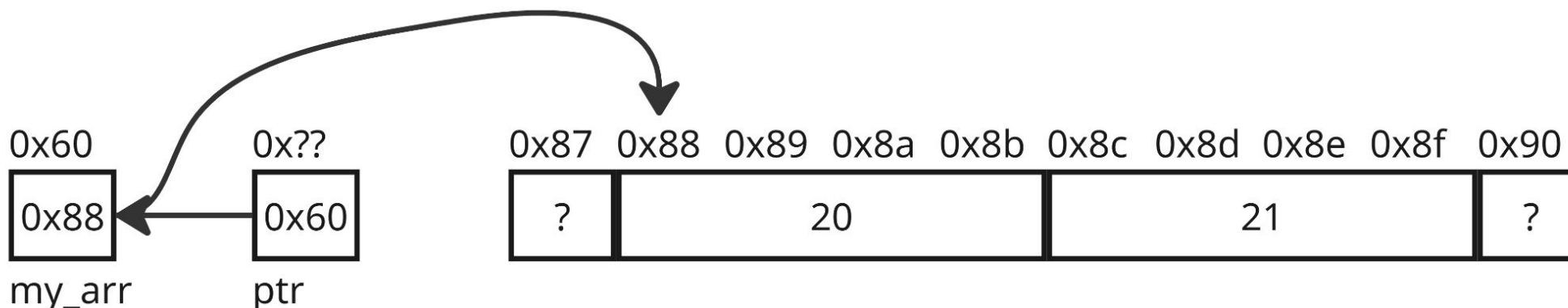
- What if we wanted to change what `my_arr` points to?
- Let's first set it up (using C++ **new** to skip **malloc**)

```
int main() {

    int32_t *my_arr = new int32_t[] {20, 21};

    int32_t **ptr = &my_arr;

    cout << my_arr << endl;    // 0x88
    cout << *my_arr << endl;   // 20
    cout << ptr << endl;       // 0x60
    cout << *ptr << endl;      // 0x88
    cout << **ptr << endl;     // 20
}
```



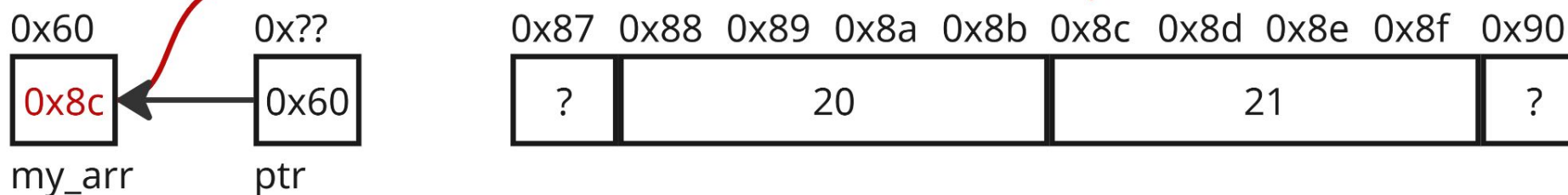
Pointer to a pointer!

- We can modify the value in **my_arr** via ***ptr**
- We can then access the data as if it was the first element of the array, or by double-dereferencing **ptr**

```
int main() {
    int32_t *my_arr = new int32_t[] {20, 21};
    int32_t **ptr = &my_arr;
    cout << my_arr << endl;    // 0x88
    cout << *my_arr << endl;   // 20
    cout << ptr << endl;       // 0x60
    cout << *ptr << endl;      // 0x88
    cout << **ptr << endl;     // 20

    *ptr += 1;

    cout << my_arr << endl;    // 0x8c
    cout << *my_arr << endl;   // 21
    cout << my_arr[0] << endl; // 21
    cout << ptr << endl;       // 0x60
    cout << *ptr << endl;      // 0x8c
    cout << **ptr << endl;     // 21
}
```



Your turn!

- Take some time to think about what this program does
- Draw a diagram and work it out

```
void foo(int32_t* x, int32_t* y, int32_t* z) {  
    x = y;  
    *x = *z;  
    *z = 37;  
}
```

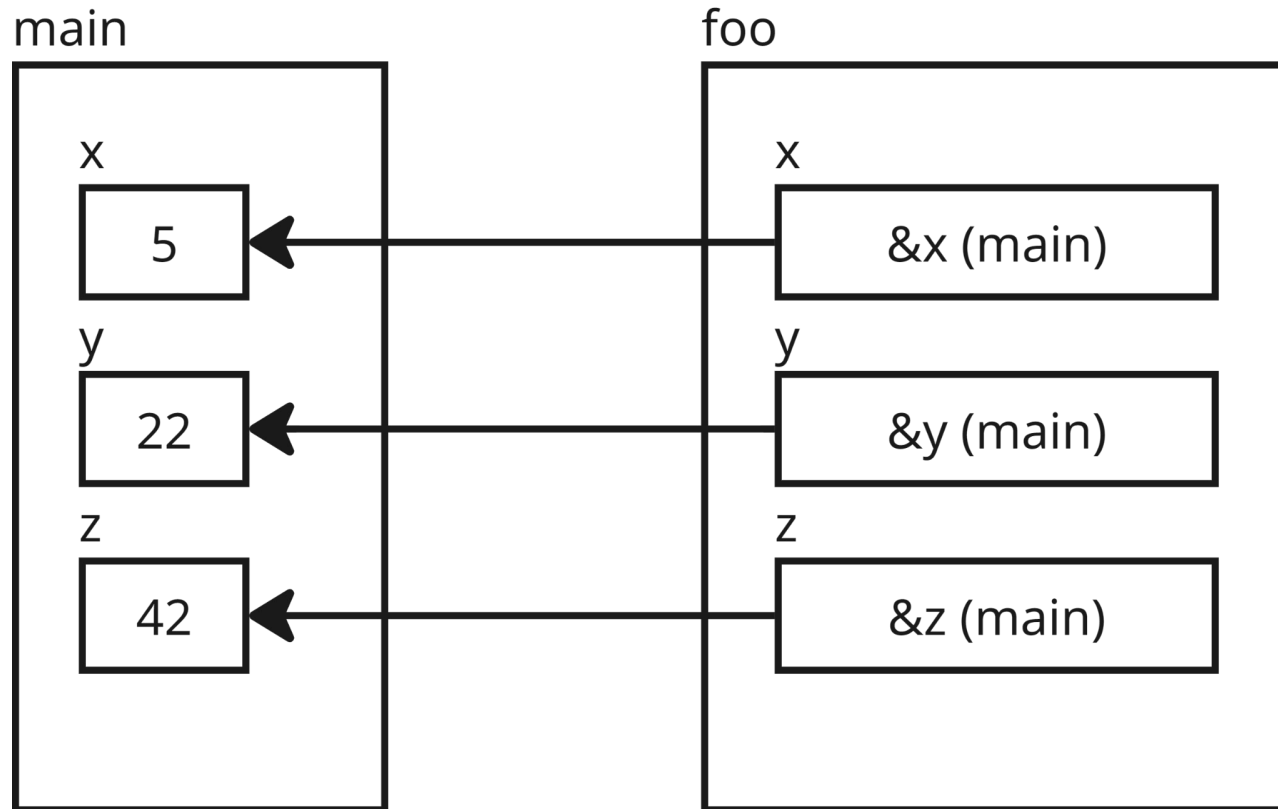
```
int main(int argc, char* argv[]) {  
    int32_t x = 5, y = 22, z = 42;  
    foo(&x, &y, &z);  
    printf("%d, %d, %d\n", x, y, z);  
    return EXIT_SUCCESS;  
}
```

Answer

- Take some time to think about what this program does
- Draw a diagram and work it out
- **Ans: "5, 42, 37"**

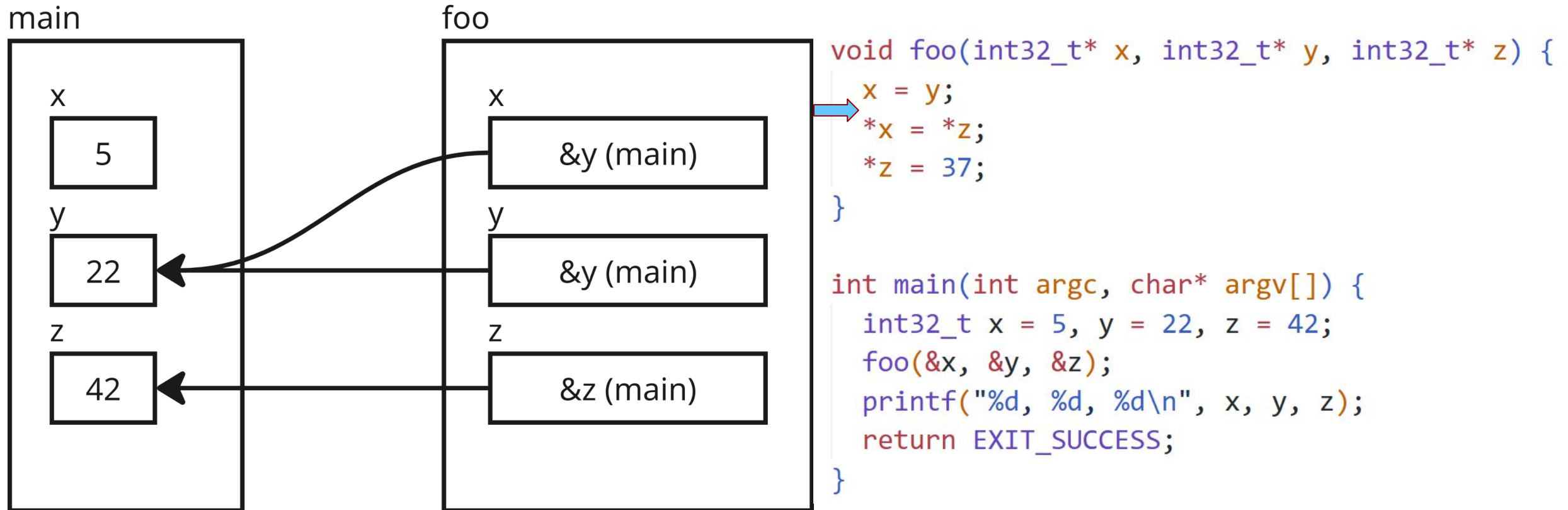
```
void foo(int32_t* x, int32_t* y, int32_t* z) {  
    x = y;  
    *x = *z;  
    *z = 37;  
}  
  
int main(int argc, char* argv[]) {  
    int32_t x = 5, y = 22, z = 42;  
    foo(&x, &y, &z);  
    printf("%d, %d, %d\n", x, y, z);  
    return EXIT_SUCCESS;  
}
```

Justifications

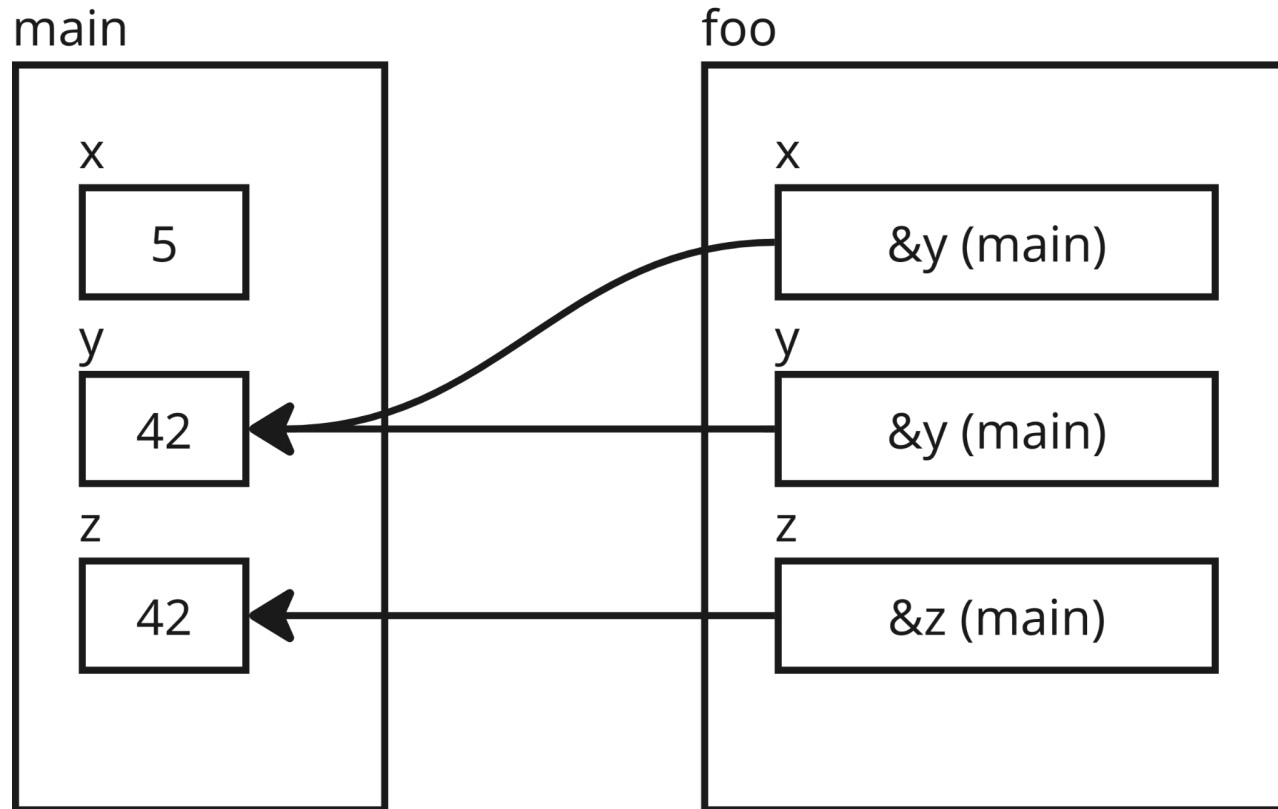


```
void foo(int32_t* x, int32_t* y, int32_t* z) {  
    x = y;  
    *x = *z;  
    *z = 37;  
}  
  
int main(int argc, char* argv[]) {  
    int32_t x = 5, y = 22, z = 42;  
    foo(&x, &y, &z);  
    printf("%d, %d, %d\n", x, y, z);  
    return EXIT_SUCCESS;  
}
```

Justifications

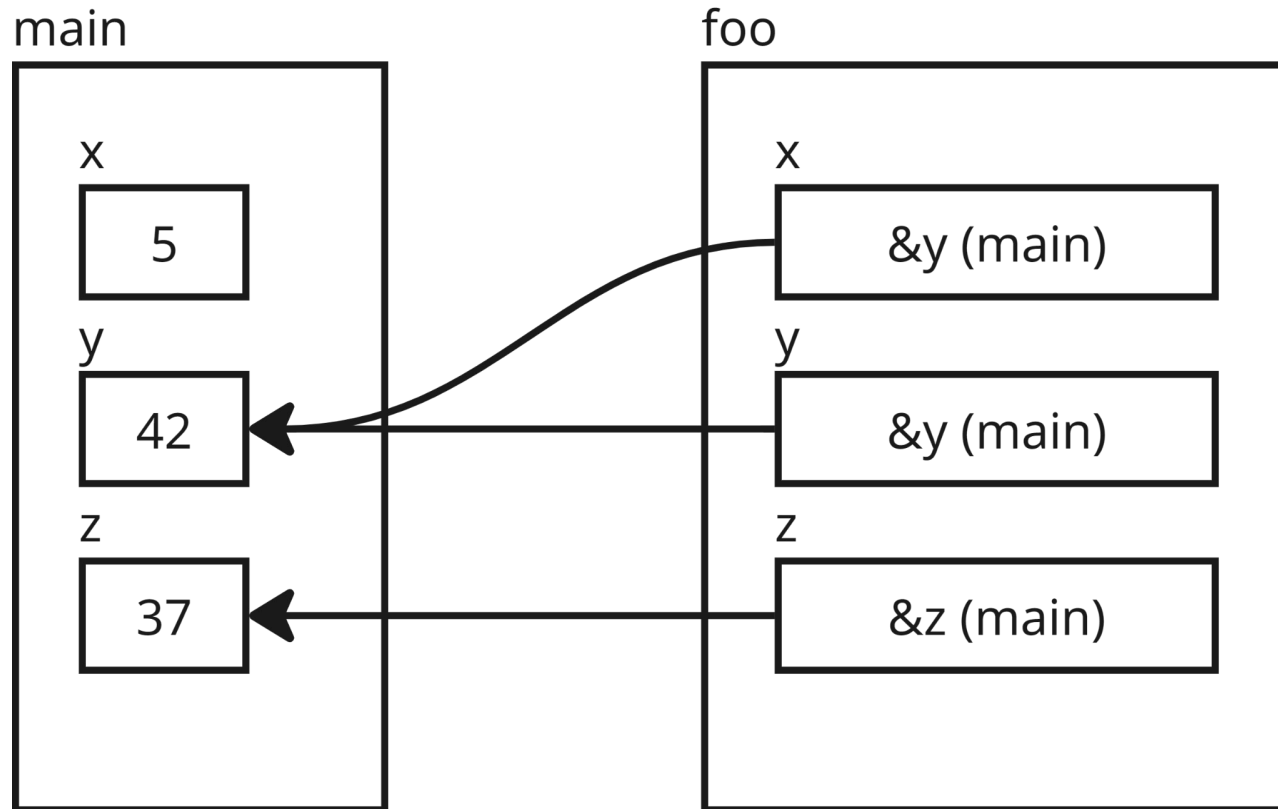


Justifications



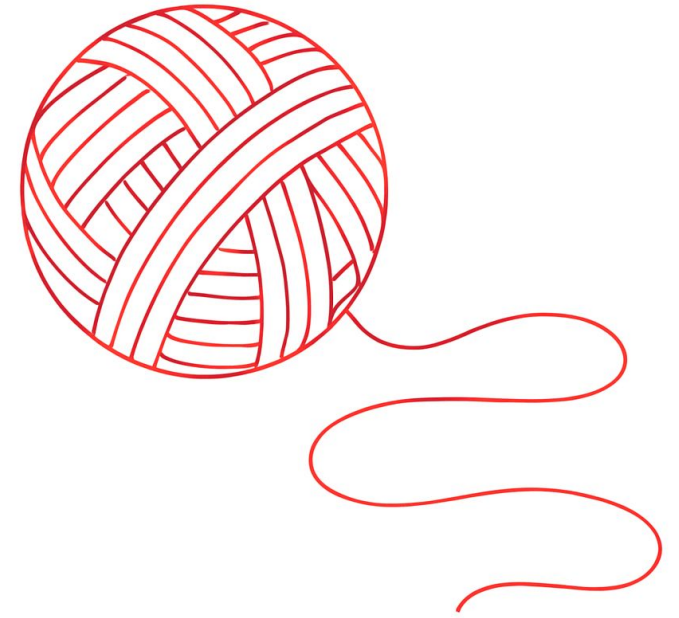
```
void foo(int32_t* x, int32_t* y, int32_t* z) {  
    x = y;  
    *x = *z;  
    *z = 37;  
}  
  
int main(int argc, char* argv[]) {  
    int32_t x = 5, y = 22, z = 42;  
    foo(&x, &y, &z);  
    printf("%d, %d, %d\n", x, y, z);  
    return EXIT_SUCCESS;  
}
```

Justifications

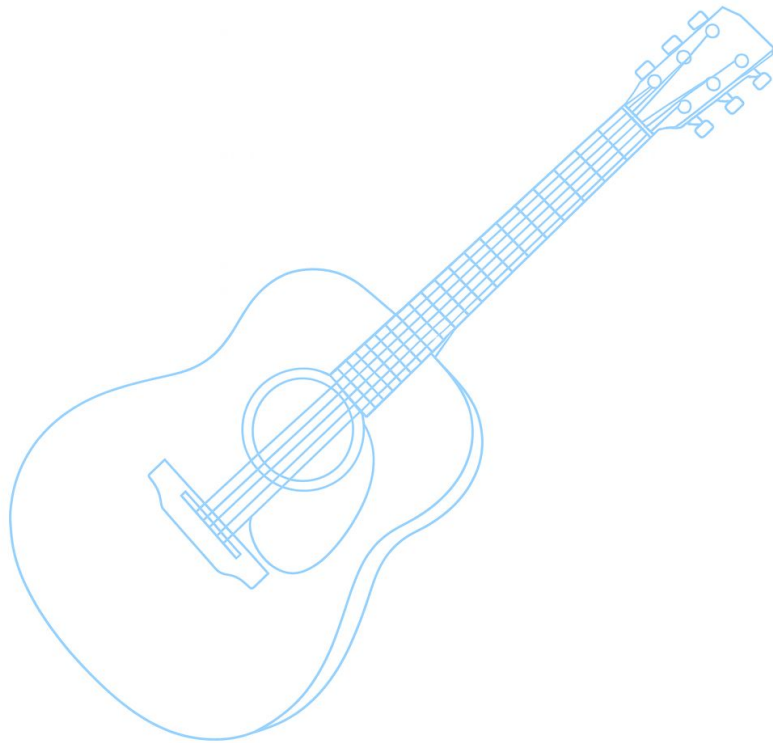


```
void foo(int32_t* x, int32_t* y, int32_t* z) {  
    x = y;  
    *x = *z;  
    *z = 37;  
}
```

```
int main(int argc, char* argv[]) {  
    int32_t x = 5, y = 22, z = 42;  
    foo(&x, &y, &z);  
    printf("%d, %d, %d\n", x, y, z);  
    return EXIT_SUCCESS;  
}
```



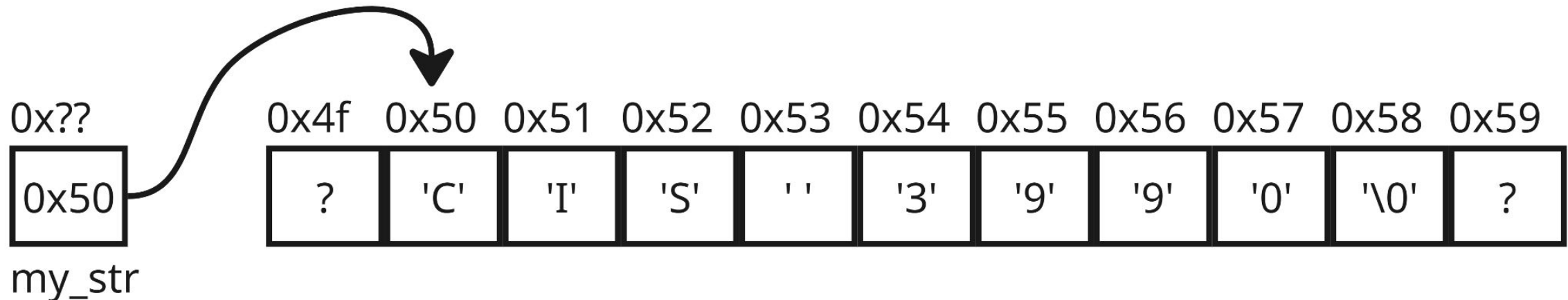
C Strings



C Strings

- They are null-terminated arrays of characters, meaning the `'\0'` character marks the end and must be present
- Remember to always allocate space for it!
- Segfault if you don't (in the good case)

```
char my_str[9] = "CIS 3990";
```



String literals can be Read-Only

- If a pointer to an in-code literal is used instead of array notation, the contents of the literal are stored in read-only memory

```
1  #include <iostream>
2  #include <cstdlib>
3
4  using namespace std;
5
6  int main(int argc, char* argv[]) {
7      char *my_str = "CIS 3990";
8
9      my_str[1] = '0';
10
11     return EXIT_SUCCESS;
12 }
```

```
● root@60e71a80138a:~/workspace/ta_3990/rec_code# clang++-15 rec_00.cpp -o rec_00
rec_00.cpp:7:18: warning: ISO C++11 does not allow conversion from string literal to 'char *' [-Wwritable-strings]
    char *my_str = "CIS 3990";
                   ^
1 warning generated.
⊗ root@60e71a80138a:~/workspace/ta_3990/rec_code# ./rec_00
Segmentation fault (core dumped)
```

Re-assignment bug?

- Does this compile?
- Does this run? If yes, what outcome does it produce? If not, what error occurs?
- How should we change the code?

```
void bar(char *str) {  
    str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

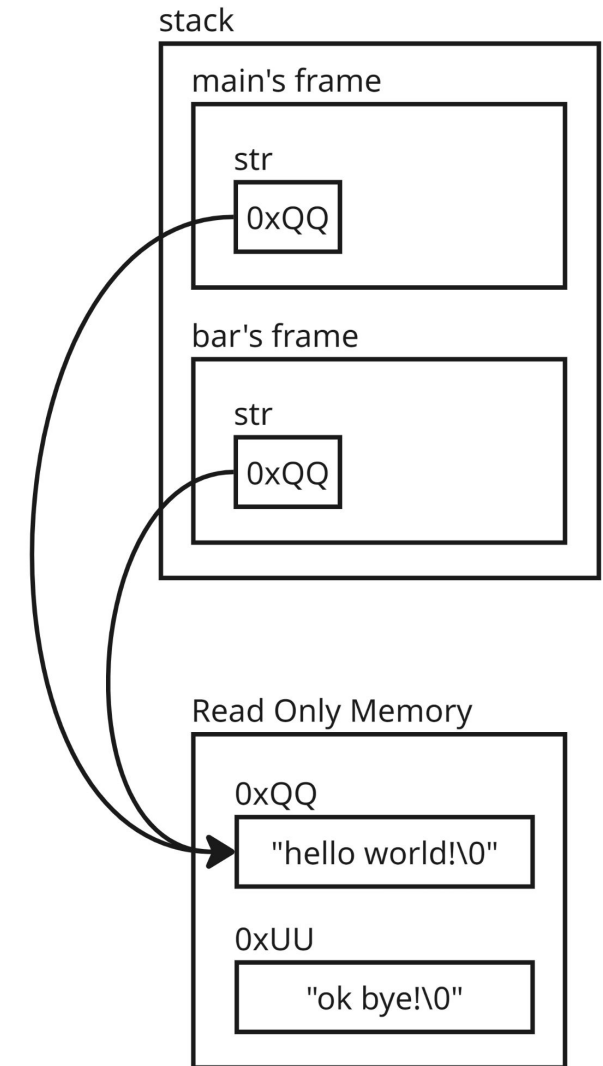
Re-assignment bug?

- Does this compile?
- **Yes**
- Does this run? If yes, what outcome does it produce? If not, what error occurs?
- **Yes, "hello world!"**
- How should we change the code?
- **Let's explain it first**

```
void bar(char *str) {  
    str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

Re-assignment bug?

```
void bar(char *str) {  
    str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

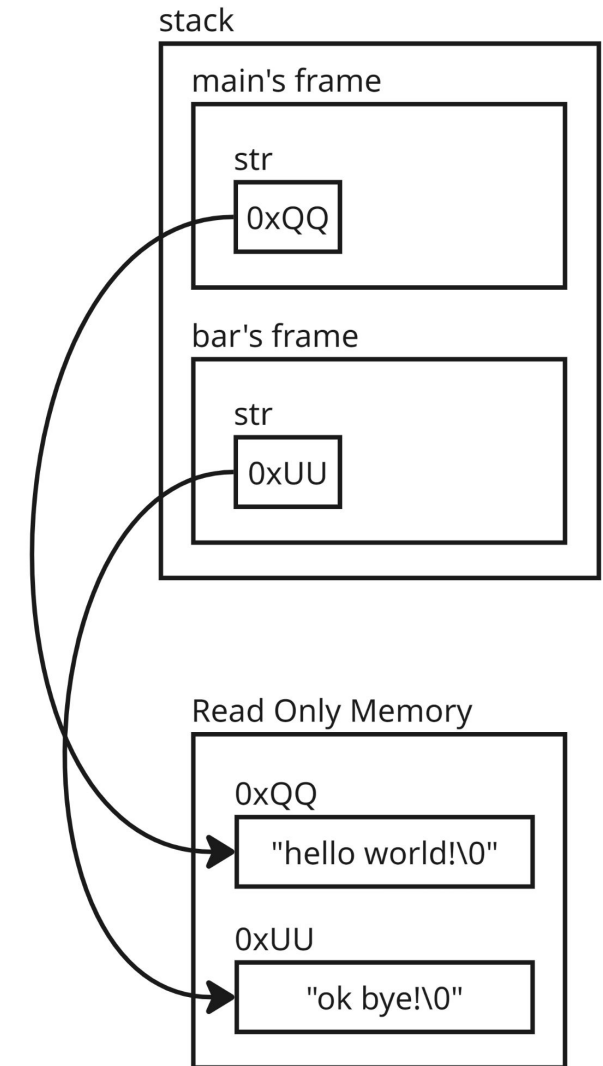


Note: 0xQQ and 0xUU are used here only to show that we're pointing to stuff in read only memory. This is not standard notation, don't think about it. What the arrows do is the important bit.

Re-assignment bug?

```
void bar(char *str) {  
    str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

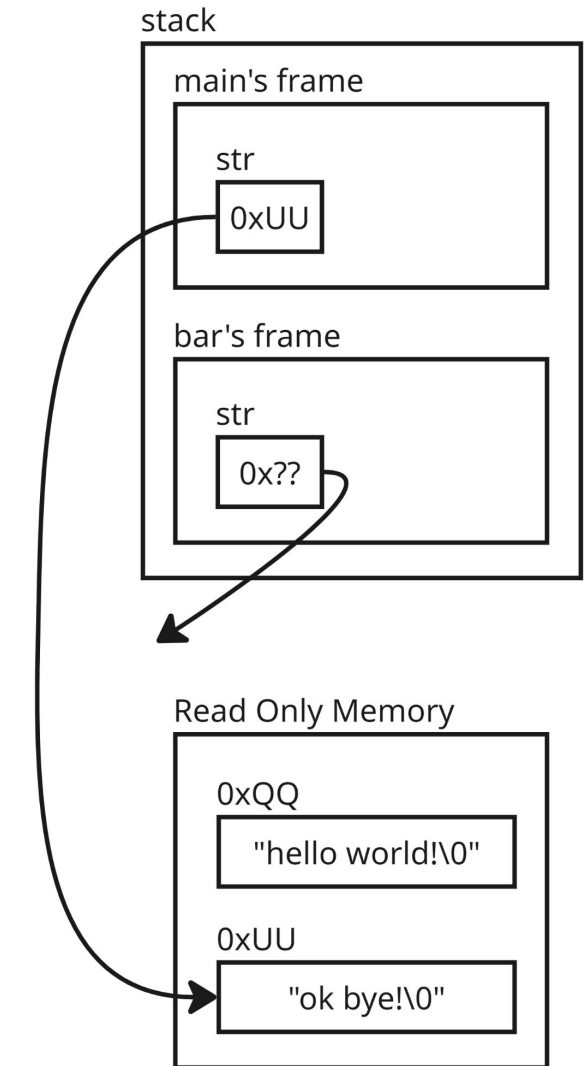
- How would we want the arrows to look at the end of bar?



Re-assignment bug?

```
void bar(char *str) {  
    str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

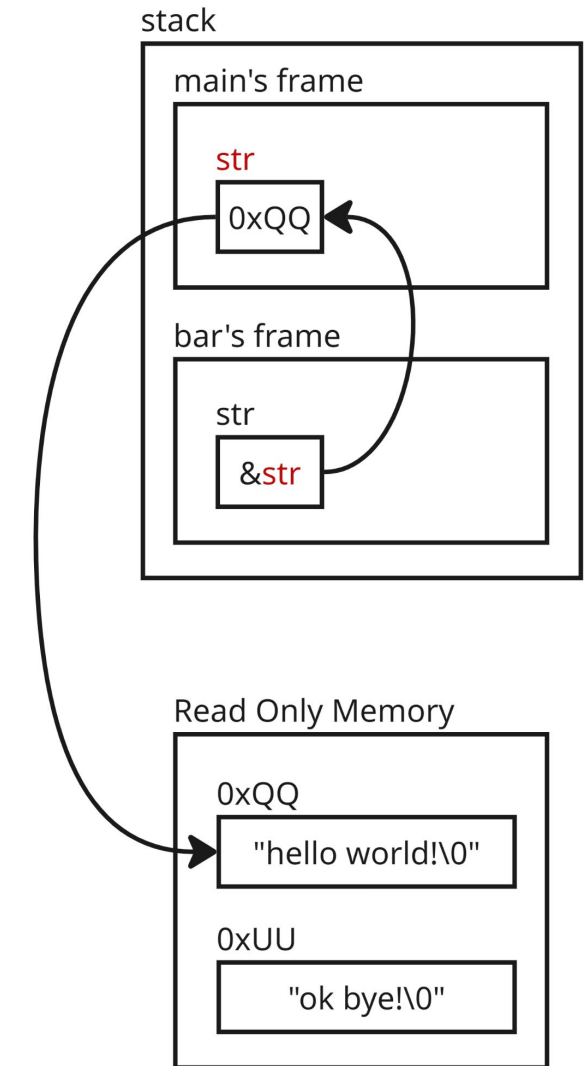
- In order to modify **main's str** within **bar**, is the value of **str** enough?



Re-assignment fix!

```
void bar(char **str) {  
    → *str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(&str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

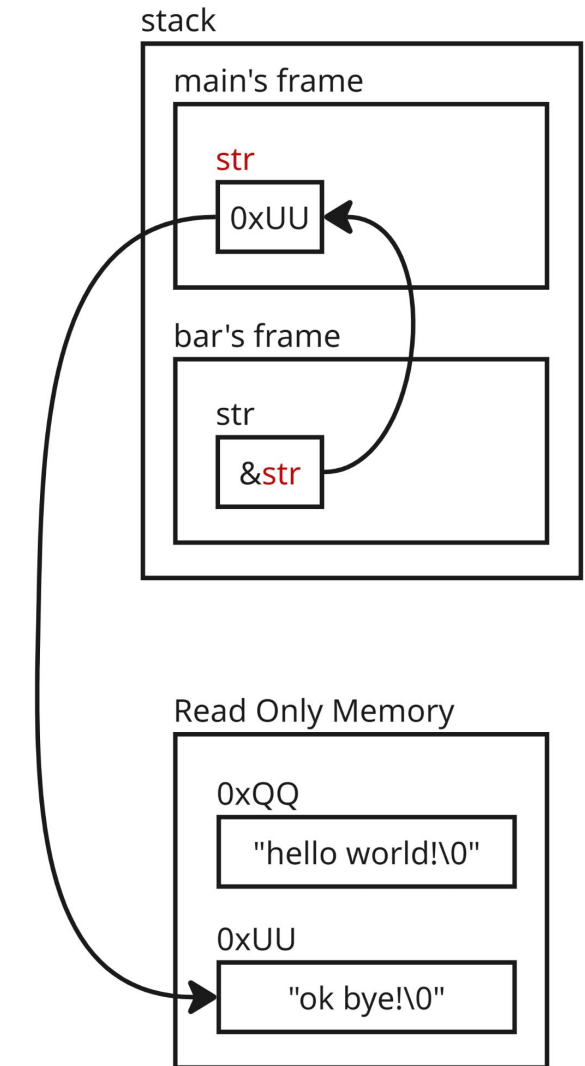
- We actually need the address of **main's str**, so we can change the value stored by **main's str**



Re-assignment fix!

```
void bar(char **str) {  
    *str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(&str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

- We actually need the address of **main's str**, so we can change the value stored by **main's str**



Output parameters

- A pointer parameter used to store output in a location specified by the caller
- Amazingly useful when a function needs to produce multiple results/ place results at a specific location

```
void bar(char **str) {  
    *str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    → char *str = "hello world!";  
    bar(&str); ←  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```

Output parameters

- **main's str** is the pointer we need to change
- We give **bar** the address of **main's str**
- **bar's str** stores the address to **main's str**, and is dereferenced in order to change **main's str** from within **bar**

```
void bar(char **str) {  
    *str = "ok bye!";  
}  
  
int main(int argc, char *argv[]) {  
    char *str = "hello world!";  
    bar(&str);  
    printf("%s\n", str); // should print "ok bye!"  
    return EXIT_SUCCESS;  
}
```



Ed practice

CIS 3990 – Ed Lessons



Lessons

 Search lessons

 Refresh

 New Module

 Import Lesson

 New Lesson

Lecture 00



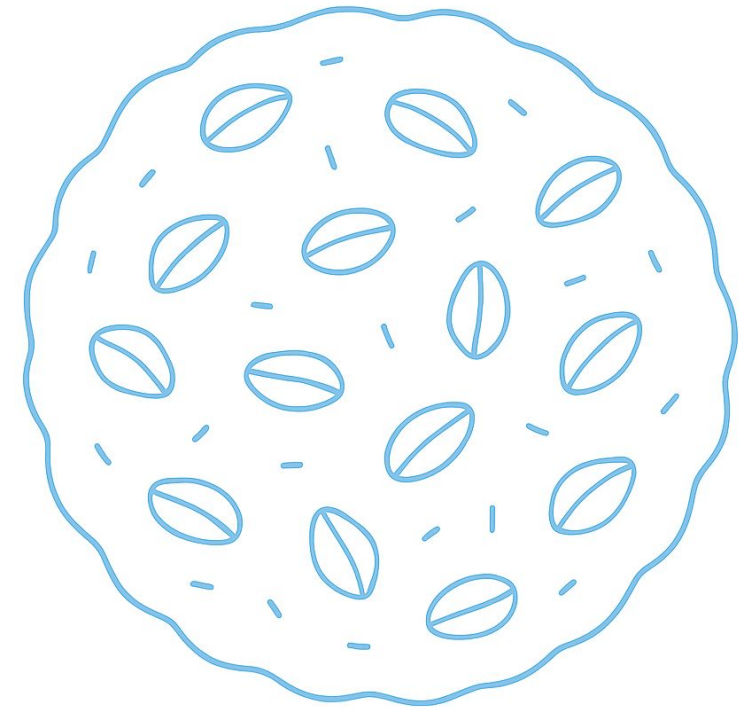
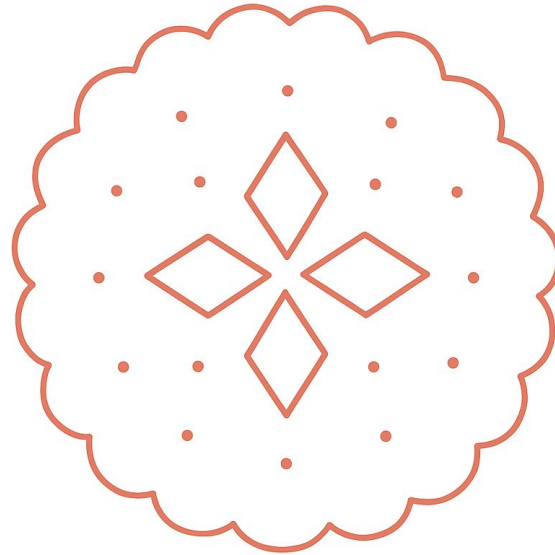
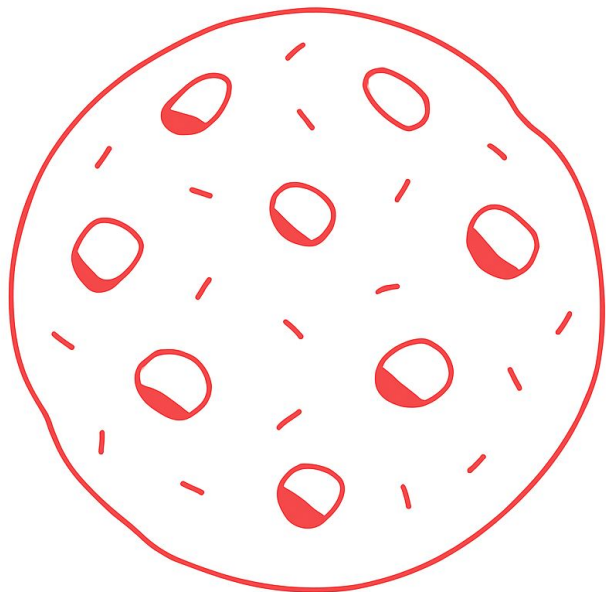
• Pointers and Memory



 C Programming Refresher



AMA & Cookies





That's all Folks!