would you rather give a 30 min talk in front of 150 people? Or write a 20 page paper?

Talk - 75%

Paper - 25%

clicker.cs.illinois.edu

Q1

~Code~
340

CS 340

C without the ++ (0b10)

← wed craft night!

Updates

1. MP 0, 1 are out
2. HW 1 is out today
3. Exam 2nd chance (now on following Thursday)

C without the C++ 0b10

LGs:

- Be able to read and understand C code
- Be able to write C code from scratch

1. Review
2. Arrays
3. C-Strings

Is there a memory error?

```
5   typedef struct node {
6     struct node *left;
7     struct node *right;
8     int datum;
9   } node;
10
11  typedef struct bst {
12    struct node *root;
13  } bst;
14
15  void init_bst(bst *self){
16    self->root = NULL;
17  }
```

```
int main() {
    bst b;
    init_bst(&b);
    return 0;
}
```

clicker.cs.illinois.edu

Q2

~Code~
340

No, no malloc

What would go in the box?

```
4   typedef struct food {
5       int amount;
6       int *age;
7   } food;
8
9   int main(){
10      int x = 5;
11      food fd;
12      fd. amount = 2;
13      fd[redacted]age = &x;
14      food *fd_p = &fd;
15      *(fd_p->age) = 7;
16  }
```

clicker.cs.illinois.edu

Q3

~Code~
340

fd is not a pointer

• dot

Arrays

What is the difference between an array and a vector?

60% – some idea

40% – confident

clicker.cs.illinois.edu

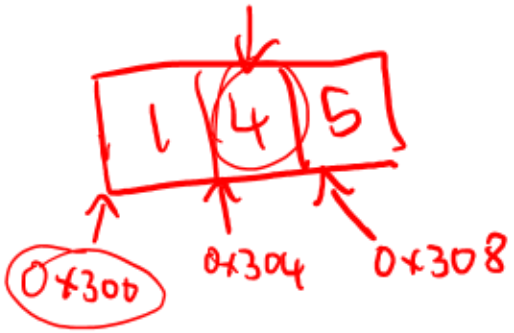
Q4

~Code~
340

[Review] What is an array?

Contiguous memory accessed through an address

```
int arr[3] = {1, 4, 5};
```



- can't change size

- access through [1] = 4

```
arr[1];   *(arr+1);
```

Pointer Math

```
int x = 4;
int *ptr = &x;
ptr = ptr + 1;
```

0x300 + 1 * sizeof(int)

*ptr; ← bad!

→ sizeof(*ptr) ← int

1 byte → Void * / Char *

Arrays and Pointer Math

```
int arr[3] = {1, 2, 3};
```

`*arr;` → 1 ← `arr[0]`

`arr` = 0x500

`*(arr+2)` → `arr[2]`

0x500 + 2 ints

8 ⇒ *(0x508) ← 3

Would these print the same or different things?

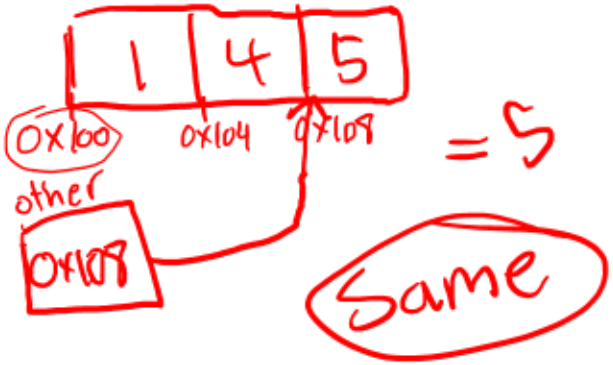
clicker.cs.illinois.edu

Q5

~Code~
340

```
15  int main() {
16      int arr[3] = {1, 4, 5};
17      int *other = arr + 2;
18      printf("%d\n", *other);
19  }
```

0x100 + 2 * sizeof(int)
8 4
= 0x108




```
15  int main() {
16      int arr[3] = {1, 4, 5};
17      int *other = arr + 2;
18      printf("%d\n", arr[2]);
19  }
```

0 1 2

5

If line 17 prints "0x13000", what would line 19 print?

clicker.cs.illinois.edu

Q6

~Code~
340

```
15  int main() {
16      int arr[3] = {1, 4, 5};
17      printf("%#x\n", arr);
18      int *other = arr + 2;
19      printf("%#x\n", other);
20  }
```

0x13000

0x13008

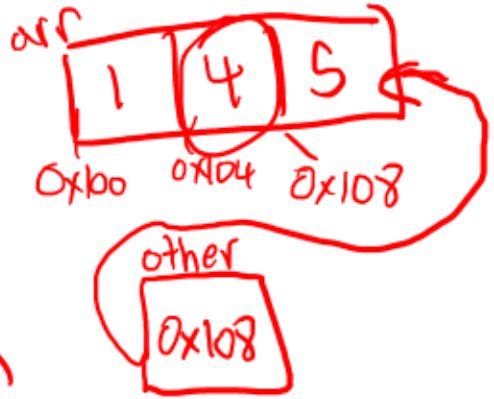
This compiles! What do you think would print?

clicker.cs.illinois.edu

Q7

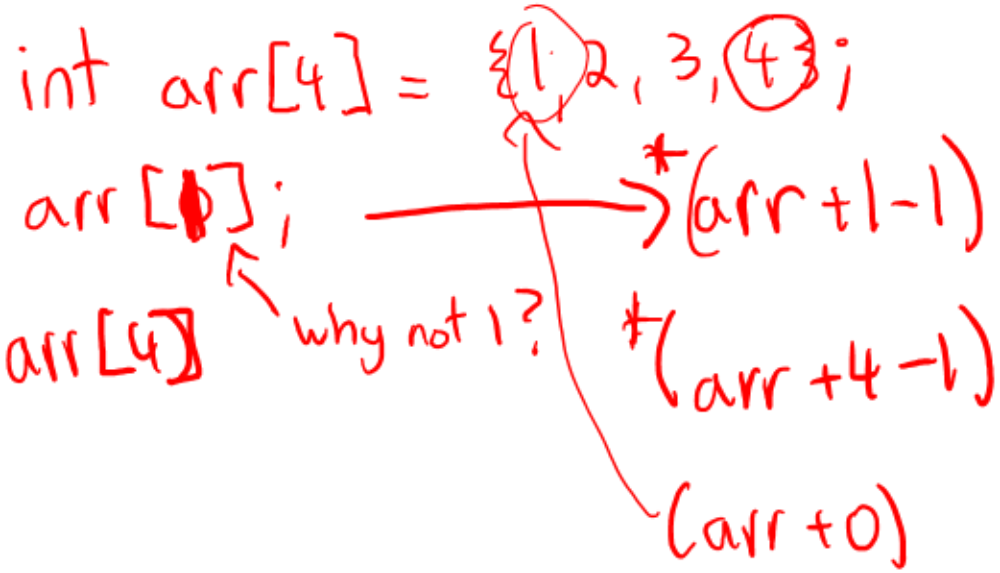
~Code~
340

```
15  int main() {
16      int arr[3] = {1, 4, 5};
17      int *other = arr + 2;
18      printf("%d\n", other[-1]);
19  }
```



*(other-1);
*(0x104) → 4

Explain to your neighbor, why we index starting at 0 in C/C++?



How does the computer know what type something is in memory?

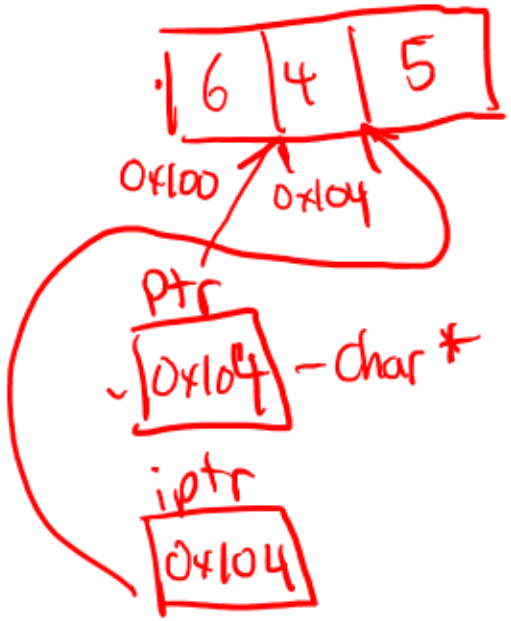
It doesn't!

- `int arr[3] = {6, 4, 5};`
- `char *ptr = (char*)arr;`

`->ptr = ptr + 4;` (4 ↑ sizeof(char))

`int *iptr = (int*)ptr;`

`printf("%d\n", *iptr);` → 4

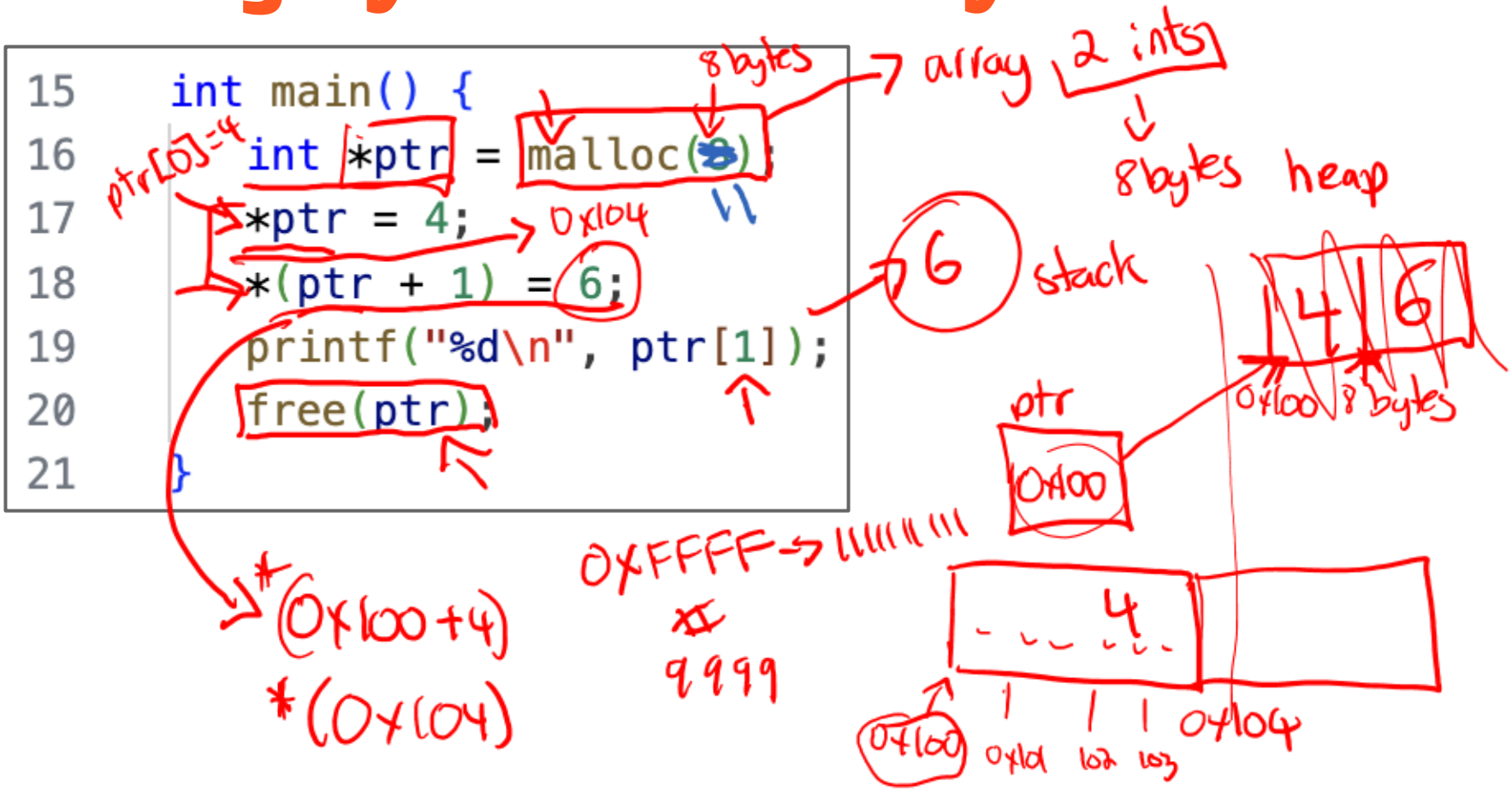


Big Ideas

1. Pointer math increases a pointer by multiples of sizeof(*ptr).
2. Arrays are contiguous bytes that can be accessed through pointer math.
3. Bytes are bytes... the type of the variable determines how the bytes are interpreted.

Dynamic Memory

Adding Dynamic Memory



Challenge! What prints?

```
15  int main() {
16      int x = 5;
17      int *ptr = malloc(8);
18      *ptr = x;
19      ptr[1] = x;
20      int** other = (int**)ptr;
21      *other = &x;
22      printf("%d\n", **other);
23      free(ptr);
24  }
```

```
//allocates size bytes on the heap and returns a
//pointer to that memory location on the heap.
void *malloc(size_t size);

//frees the memory at ptr from the heap
void free(void *ptr);

//allocates num * size bytes on the heap and returns
// a pointer to that memory location on the heap
void *calloc(size_t num, size_t size);

//changes the memory allocated at ptr to now be size.
//It copies over previous values.
void *realloc(void *ptr, size_t size);
```

num * size = bytes

4

sizeof(int)

Calloc and free Example

```
15  int main() {
16      int *arr = calloc(5, sizeof(int));
17      arr[4] = 5;
18      free(arr);
19  }
```

What will happen?

```
15  int main() {
16      int *arr = calloc(2, sizeof(char));
17      *arr = 4;
18      printf("%d\n", *arr);
19      free(arr);
20  }
```

clicker.cs.illinois.edu

Q8

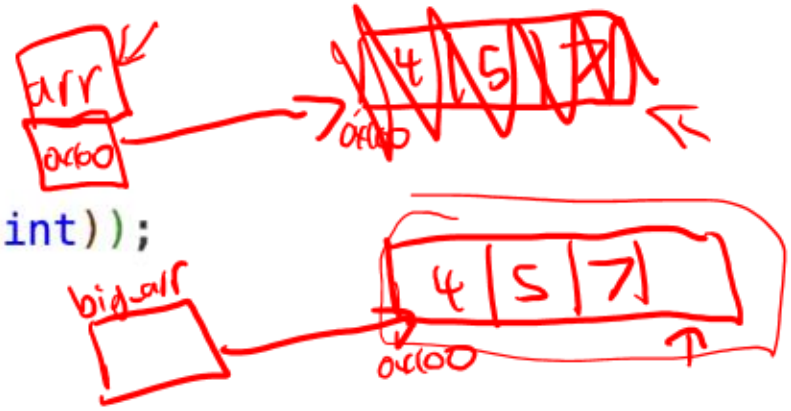
~Code~
340

realloc

```
void *realloc(void *ptr, size_t size);
```

realloc Example

```
15  int main() {
16      int *arr = calloc(3, sizeof(int));
17      arr[0] = 4;
18      arr[1] = 5;
19      arr[2] = 7;
20      int *bigger_arr = realloc(arr, sizeof(int)*4);
21      bigger_arr[3] = 10;
22      free(bigger_arr);
23  }
```



C-Strings

char - 1 byte = 8 bits = 0-255

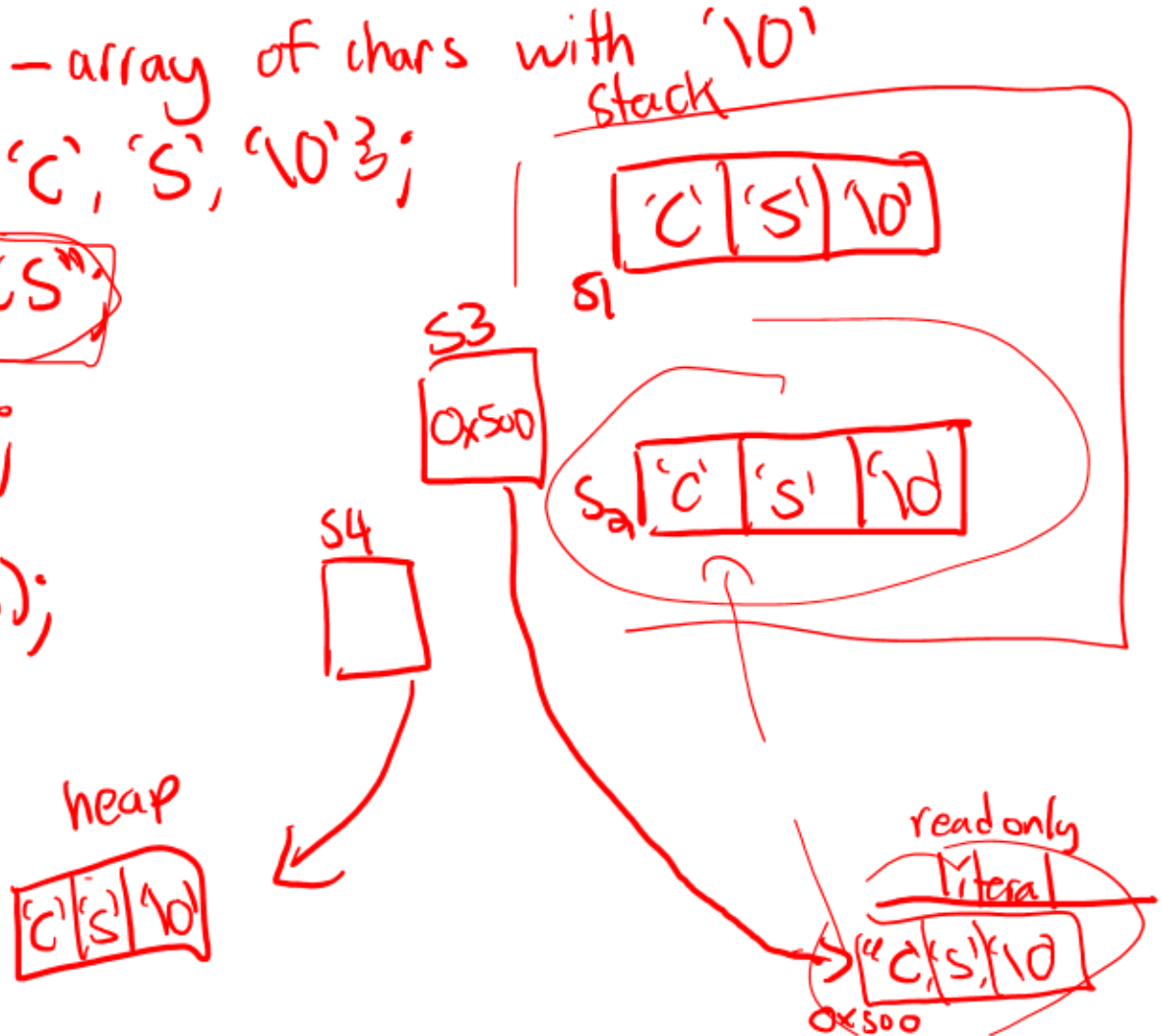
ascii - maps number to a character

'a' → 97

'A' → 65

C-Strings/char* – array of chars with '\0'

```
char s1[3] = {'C', 'S', '\0'};
char s2[3] = "CS";
char *s3 = "CS";
char* s4 = malloc(3);
strcpy(s4, "CS");
```



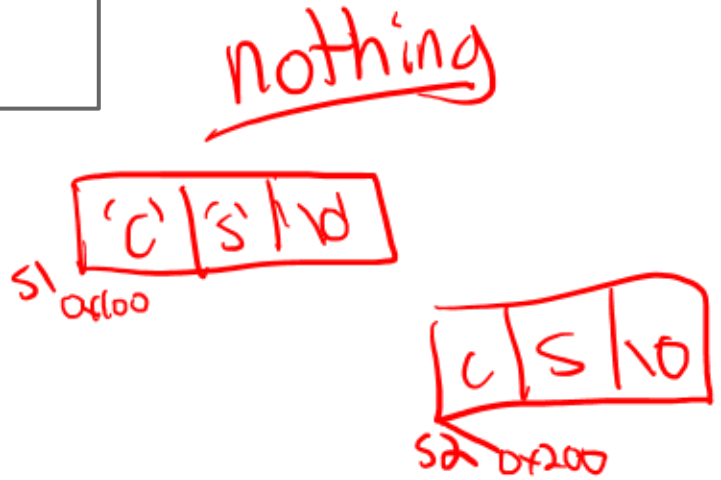
What prints?

```
15  int main() {
16      char s1[3] = {'C', 'S', '\0'};
17      char s2[3] = {'C', 'S', '\0'};
18      if (s1 == s2){
19          printf("yay");
20      }
21  }
```

clicker.cs.illinois.edu

Q9

~Code~
340



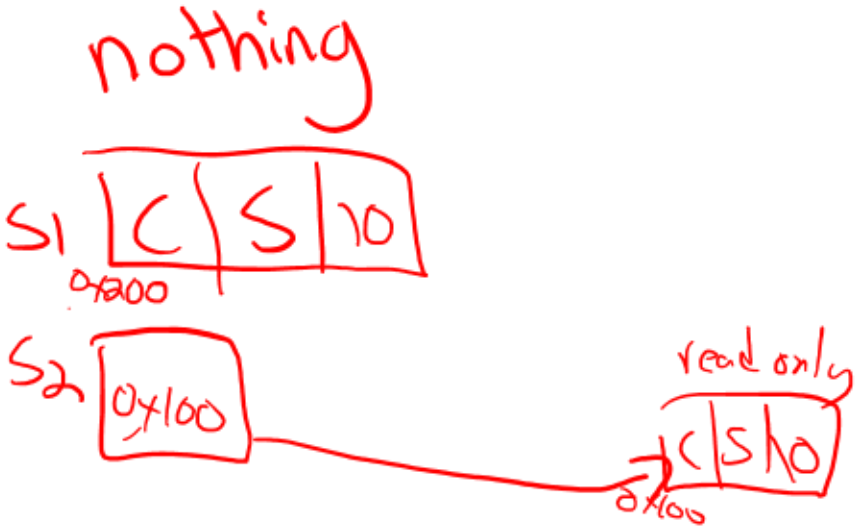
What prints?

```
15  int main() {
16      char s1[3] = "CS";
17      char *s2 = "CS";
18      if (s1 == s2){
19          printf("yay");
20      }
21  }
```

clicker.cs.illinois.edu

Q10

~Code~
340



What prints?

```
15  int main() {
16      char *s1 = "CS";
17      char *s2 = "CS";
18      if (s1 == s2){
19          printf("yay");
20      }
21  }
```

clicker.cs.illinois.edu

Q11

~Code~
340

yay

#include <string.h>

size_t strlen(char *str)

char *strcpy(char *dest, const char *src)

int strcmp(const char *str1, const char *str2)

What is the issue with the top code snippet?

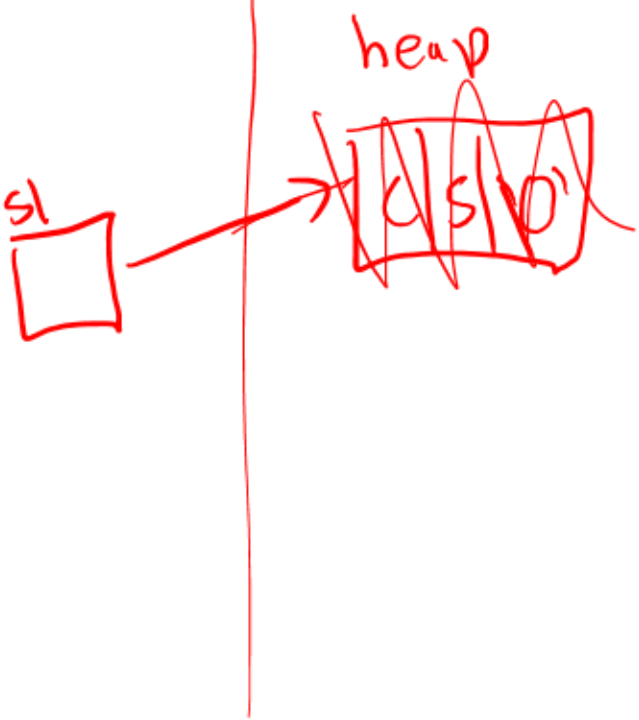
```
15  int main() {
16      char *s1 = malloc(sizeof(char)*3);
17      s1 = "CS";
18      free(s1);
19  }
```

Bad

```
15  int main() {
16      char s1[3] = "CS";
17  }
```

Solution!

```
15    #include <string.h>
16
17    int main() {
18        char *s1 = malloc(sizeof(char)*3);
19        strcpy(s1, "CS");
20        free(s1);
21    }
```



int printf(const char * format, args);

1. %d or %i: for integers
2. %#x: for 0xHEX_NUMBER
3. %c: for characters
4. %s: for C-strings
5. ...more

```
13 int main() {
14     int x = 5;
15     int *ptr = &x;
16     char *str = "hi";
17     printf("will print: %i, %#x, %s", x, ptr, str);
18 }
```

input

```
will print: 5, 0xd88ae92c, hi
```

Arrays in Functions

C without the C++ 0b10

LGs:

- Be able to read and understand C code
- Be able to write C code from scratch

1. Review
2. Arrays
3. C-Strings