



# **W25 CS 136**

# **Midterm Review**

# Content Covered in the Midterm:

Functional  
Programming

Imperative  
Programming

C Memory Model

Pointers

Arrays and Strings

# Short Answer (5 minutes)

- 1) What's the difference between pass by reference vs pass by value?
- 2) Difference between `const int *p` and `int * const p`
- 3) Where is a string literal stored?
- 4) What is the difference between imperative and functional programming?  
Which one does C fit into? How about Racket?

## 1) What's the difference between pass by reference vs pass by value?

Pass by value sends a copy of the data, pass by reference sends the address. If you do pass by reference, it's possible to mutate the original value of the data that was given to the function.

## 2) Difference between `const int *p` and `int * const p`

`const int *p` is a pointer to a constant integer. You can't mutate the data pointed to by `p`, but can change what `p` is equal to.

`int * const p` is a constant pointer to an integer. You can mutate the data pointed to by `p`, but can't change what `p` is equal to.

### 3) Where is a string literal stored?

Read only data

### 4) What is the difference between imperative and functional programming? Which one does C fit into? How about Racket?

Functional Programming: Functions are pure (no side effects), functions only return values, return values only depend on argument values, only constants are used

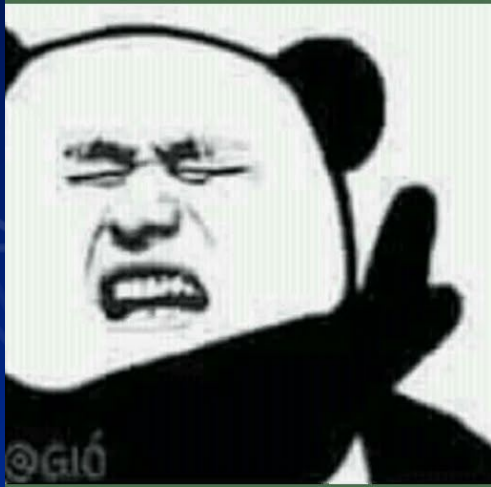
Procedural Programming: functions may be "impure", variables and constants are used, and there are side effects

We've used both a functional and procedural programming paradigm for C.  
We've only used a functional paradigm for Racket.



# Linear Algebra

Instead of studying for MATH136, many students are busy playing league of legends. Help them create a program to perform matrix multiplication to aid their studying and lock in!



Bro's Math136 grades  
♥♥♥♥♥

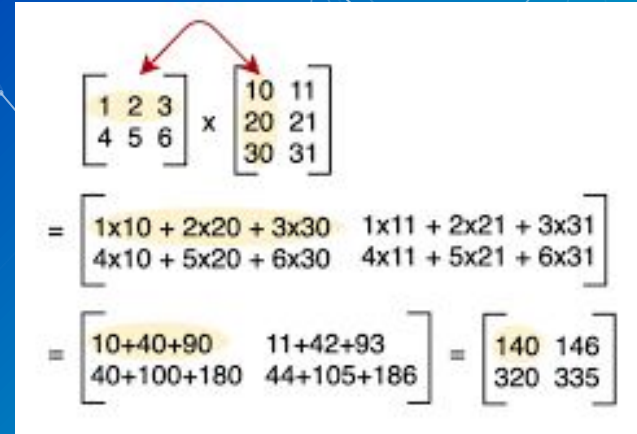


# linalg (15 minutes)

Write a function `matrix_mult` that takes in 3 matrices, each represented by a struct array. If the matrix multiplication between `m1` and `m2` is possible, the function will mutate `matrix_product` and its fields to be the result of the matrix multiplication.

```
// matrix_mult(m1, m2, matrix_product) is a function that takes in 2 matrices
// and if possible, will perform matrix multiplication on them. the values of
// the multiplication will be stored in matrix_product's values, and the height
// and width of matrix product will be stored in matrix_product's fields
// respectively. returns true if the matrix multiplication is successful.
//
// Requires:
//   all pointers are not null
// Effects:
//   May modify matrix_product

bool matrix_mult(const struct matrix *m1, const struct matrix *m2,
                 struct matrix *matrix_product) {
    // Here so that the code will compile.
    return false;
}
```


$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$
$$= \begin{bmatrix} 1 \times 10 + 2 \times 20 + 3 \times 30 & 1 \times 11 + 2 \times 21 + 3 \times 31 \\ 4 \times 10 + 5 \times 20 + 6 \times 30 & 4 \times 11 + 5 \times 21 + 6 \times 31 \end{bmatrix}$$
$$= \begin{bmatrix} 10 + 40 + 90 & 11 + 42 + 93 \\ 40 + 100 + 180 & 44 + 105 + 186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

# Enumeration

The season just started and you are on a journey to reach Rank 1 in League of Legends. But there is something stopping you; Riot has implemented a new CAPTCHA that requires you to do some counting before you can log in.

- Given an integer  $n$ , count how many times the digit '1' appears in all numbers from 0 to  $n$ .

There's a slow method, which if you use you are going to login too late and someone else will get rank 1 before you.

But if you use the faster method you will be able to hit rank 1 first! Will your determination to reach rank 1 transfer to your programming abilities?



## Qualifying question

Just to prove you are a human, please answer the following math challenge.

Q: Calculate:

$$\left. \frac{\partial}{\partial x} \left[ 5 \cdot \sin \left( 5 \cdot x + \frac{\pi}{2} \right) \right] \right|_{x=0}$$

A:

mandatory



# countones (20 min)

Implement `count_ones(n)`, which returns the total number of digit 1 appearing in all non-negative integers less than or equal to `n`. You must use recursion.

Try both the easy and hard implementation!

```
// Given an integer n, count the total number of digit 1 appearing
//   in all non-negative integers less than or equal to n.
// Use recursion for this question
// Easy implementation, pass all asserts except the last one
// Hard implementation, pass all asserts including the last one
//   (you will need an efficient solution)
// Requires: n is non-negative
// Examples:   count_ones(4) => 1
//             count_ones(12) = 6
int count_ones(int n) {
    return 0;
}
```

```
int main(void) {
    assert(count_ones(0) == 0);
    assert(count_ones(1) == 1);
    assert(count_ones(10) == 2);
    assert(count_ones(11) == 4);
    assert(count_ones(19) == 12);
    assert(count_ones(20) == 12);
}
```

# Trade Trouble

Recently, many players in the NBA were traded, including AJ Johnson (#77), Reggie Jackson (#7), and Luka Doncic (#77)

Bob, a Waterloo basketball player, noticed that all these players had a jersey number divisible by 7. He wears #136, since his favorite course CS136, and wonders whether his jersey is also divisible by 7.



# Divisibility by 7 method

Luckily, Bob knows a method to check if a number is divisible by 7:

- 1) Take the last digit and double it
- 2) Subtract it from the remaining digits
- 3) Repeat until number  $< 10$
- 4) If the reduced number is divisible by 7, so was the original

For example,  $777 \rightarrow 77 - (7 * 2) \rightarrow 63 \rightarrow 6 - (3 * 2) \rightarrow 0$  (divisible)

$1234 \rightarrow 115 \rightarrow 1$  (not divisible)

# Lukarecursion (20 min)

Implement `reduce_seven`, which reads a number, possibly separated by spaces, and returns the reduced number given by the method.

Note that "1 2 3 4" is a valid number.

```
// reduce_seven(num): Reads a number, avoiding whitespace, and reduces
//      it to a number < 10 by the following strategy:
//  1) Take the last digit and double it
//  2) Subtract it from the remaining digits
//  3) Repeat until number < 10
// Note that num has no special meaning, it's just a parameter you can use to pass
// numbers through recursive loops
// You must use recursion, without making any new functions.
// Examples: (quotations represent standard input)
//      "1 2 3 4" -> 1234 -> 123 - (4 * 2) -> 115 -> 11 - (5 * 2) -> 1
//      " 777 " -> 63 -> 0
// Effects: Reads from console
int reduce_seven(int num) {
    return 0;
}
```



# NBA All Star 2026

Many fans hated the NBA All Star game this year, so for 2026, the NBA All Star game will feature a much requested 1v1 matchup between LeBron James and his son, Bronny James.



vs





# Memory Snapshots for Arrays

(Might be different notation on the exam)

```
int main(void) {  
    int arr[3] = {1, 2, 3};  
    int *ptr = arr + 2;  
}
```

main:

arr:

[0]: 1

[1]: 2

[2]: 3 [addr\_1]

ptr: addr\_1

# Memory Snapshots for String Literals

(Might be different notation on the exam)

```
int main(void) {  
    char *c = "Bronny";  
}
```

```
1 READ-ONLY DATA:  
2  
3 [0]: 'B' [addr_1]  
4 [1]: 'r'  
5 [2]: 'o'  
6 [3]: 'n'  
7 [4]: 'n'  
8 [5]: 'y'  
9 [6]: '\\0'
```

```
10  
11  
12 STACK:  
13  
14 =====  
15 main:  
16 c: addr_1  
17
```

# LeMemory Snapshots (20 min)

Create the memory snapshot for main.c



```
5
6 struct player {
7     char *name;
8     int fg2;
9     int fg3;
10 };
11
12 const int POSSESSIONS = 20;
13
14 struct player BRONNY = {"Bronny", 37, 21};
15
16 void one_v_one(struct player *a, struct player *b)
17 {
18     printf("%s vs %s\n", a->name, b->name);
19     if (strcmp(a->name, b->name) > 0) {
20         b->name[0] = 'J';
21     } else {
22         b->name += 2;
23     }
24     // SNAPSHOT A
25     printf("%s vs %s\n", a->name, b->name);
26     a->name++;
27     char temp[7] = {0};
28     strcpy(temp, a->name);
29     *temp = 'S';
30     temp[3] = '\0';
31     printf("%s vs %s\n", temp, b->name);
32     // SNAPSHOT B
33     strcpy(b->name, temp);
34     if (a->fg2 * 2 < b->fg3 * 3) {
35         b->name[1] = 'u';
36     } else {
37         temp[1] = 'u';
38     }
39     // SNAPSHOT C
40     printf("%s vs %s\n", temp, b->name);
41 }
42
43 int main(void) {
44     char name[8] = "LeBron";
45     char *storename = name;
46     struct player LEBRON = {storename, 25, 40};
47     one_v_one(&BRONNY, &LEBRON);
48 }
```



# char\_limit (1 minute)

Implement `char_limit(str)`, which returns true if the character limit is obeyed.

```
// char_limit(str): Returns true if str is <= the character limit MAX_LEN
// requires: str is a valid string
bool char_limit(const char *str) {
    return true;
}
```



# starts\_with (5 minutes)

Implement `starts_with(a, b)`, which determines if *a* starts with sequence *b*

```
// starts_with(a, b): Returns true if a starts with b
// examples: "bod" starts with "bo", but "bo" does not start with "bod"
// requires: str a, b are valid strings
bool starts_with(const char *a, const char *b) {
    return true;
}
```

# Bad spacing

Joe compiled a list of all possible bad spacing errors, denoted in the badspaces string array.

He detects a line as having bad spacing if one of these errors appears anywhere in the line.

Additionally, if an errors happens after a `//`, it is ignored.

```
const char *badspace[6] = {"if(", "for(", "while(", "){", "else{", "}else"};  
const int BADSPACES_LEN = 6;
```

# Space\_check (10 minutes)

Implement `space_check(str)`, which returns true if there's no bad spacing issues.

```
// space_check(str): Returns false if there's an instance of any bad spacing errors denoted in badspaces
// The space checker ignores any bad spaces after a single-line comment "//"
// requires: str is a valid string
bool space_check(const char *str) {
    return true;
}
```

# Uninitialized Variables

Joe wants to detect for any lines consisting of declaring uninitialized variables. He detects these as follows:

- 1) After any initial whitespace, the line starts with "int", "char", "bool", or "struct". This is given by the types array.
- 2) The line ends with a semicolon
- 3) The line does not have an equal sign.

```
const char *types[4] = {"int", "char", "bool", "struct"};  
const int TYPES_LEN = 4;
```

# initialized (10 minutes)

Implement `initialized(str)`, which returns `false` if the string consists of declaring an uninitialized variable, based on the method given in the previous slide.

```
// initialized(str): Returns false if str meets this criteria:  
// 1) After any initial whitespace, it starts with one of the types given by the types array  
// 2) Ends with ;  
// 3) Does not contain =  
// examples: "    int a;" -> false  
//           "    int a; " -> true  
//           "int** a;" -> false  
//           "int a = 5;" -> true  
// requires: str is a valid string  
bool initialized(const char *str) {  
    return true;  
}
```



# WE DID IT!



JAYSON  
TATUM



PTS  
31

FG  
11-24

REB  
8

AST  
11

## Questions?