Midterm Review

Basics of C

Arrays

- Lists of data
- Have no concept of their own size
- Easily indexed
    - Ex: int arr[] = {1, 2, 3}; → arr[2] == 3
    - What is arr…?

Pointers

- Variables that “point” to other things
- * → dereferences a pointer
- & → gets the address of some data

```c
int main() {
    int arr[] = {10, 20, 30, 40};
    int **pptr, *ptr1, *ptr2;

    ptr1 = arr;
    ptr2 = ptr1 + 2;
    pptr = &ptr2;

    // playing with pointers
    (*pptr)--;
    **pptr += 5;

    *(ptr1 + 3) = **pptr * 2;

    // call the manipulate function with pointers
    manipulate(&ptr1, ptr2);

    // printing the modified array
    for (int i = 0; i < 4; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

```c
// function to manipulate pointer arguments
void manipulate(int **p1, int *p2) {
    (*p1)++;
    *(*p1) *= 2;

    (*p2)--;
    *p2 += 10;
}
```

What should this print?

Ans: 10, 59, 30, 50

Strings

- ALMOST identical to arrays, with some additional useful syntax and one key feature: NULL TERMINATED
- Which of these is a string?
  - char str1[] = “hi”;
  - char str2[] = {‘h’, ‘i’};
  - char* str3 = “hi”;

Struct

- Use when you want to create your own data structure
- Ex:

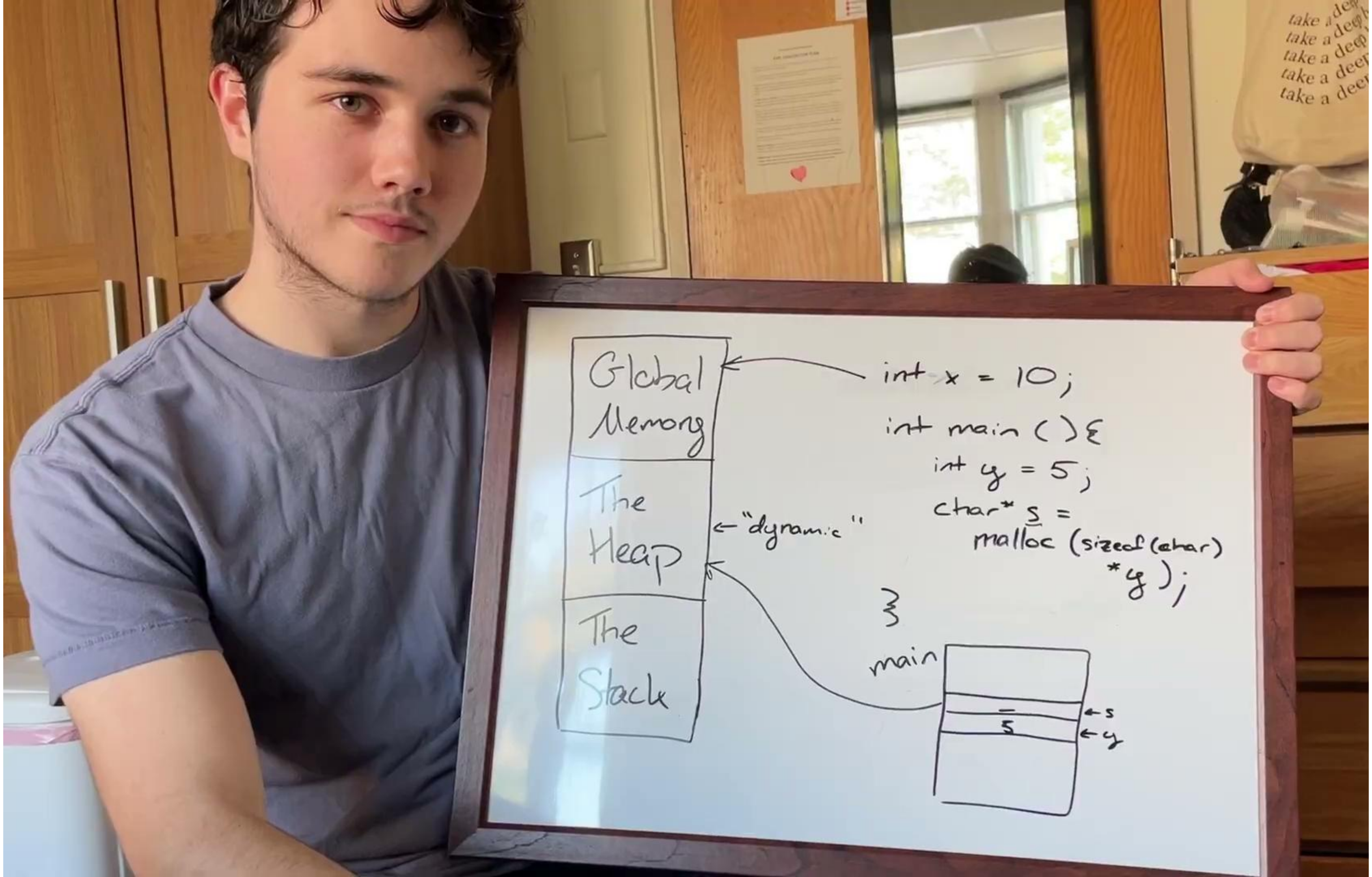
```
typedef struct node_st {
    int value;
    node_st* next;
    node_st* prev;
} Node;
```

*If you want to have the same data structure (or in this case a pointer to one) as one of your fields, you HAVE to give it a temporary name up top (ex: node_st)

C Memory

- In this class, we think of memory as giant arrays that we can store variables and programs in
- The Stack
  - Where all your functions and local variables exist
  - Once a function is returned, it gets “popped” off the stack
- The Heap
  - “Dynamic” memory → allocated during runtime
  - malloc()
- Global Memory
  - Declared outside any function
  - Can be accessed from any function
  - Any global variables exists as long as your program is running

Memory Demo
(watch on your own time)


Global Memory
The Heap
The Stack
"dynamic"
int x = 10;
int main ( ){
int y = 5;
char* s =
malloc (sizeof(char)
*y);
}
main
5
s
y

malloc() and free()

- In this class, we think of memory as giant arrays that we can store variables and programs in
- The Stack
  - Where all your functions and local variables exist
  - Once a function is returned, it gets “popped” off the stack
- The Heap
  - “Dynamic” memory → allocated during runtime
  - malloc()
- Global Memory
  - Declared outside any function
  - Can be accessed from any function
  - Any global variables exists as long as your program is running

Binary and Hexadecimal

Binary

2^1	=	2
2^2	=	4
2^3	=	8
2^4	=	16
2^5	=	32
2^6	=	64
2^7	=	128
2^8	=	256
2^9	=	512
2^{10}	=	1024

Hexadecimal

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

$16^1 = 16$
$16^2 = 256$
$16^3 = 4096$
$16^4 = 65536$

2’s Complement

- Signed integers (can represent positive or negative values)
- Negative Numbers - 1 Most Significant Bit
- Positive Numbers - 0 Most Significant Bit
- Negate binary numbers : Invert (1’s turn to 0’s and 0’s turn to 1’s) -> Plus 1

Binary Practice - 2s complement - 8 bits

Base 2:
Base 10: 95
Base 16:

Base 2:
Base 10:
Base 16: F8

Base 2: 10 1101
Base 10:
Base 16:

Base 2:
Base 10: -3
Base 16:

Binary Practice - 2s complement - 8 bits

Base 2: **0101 1111**
Base 10: 95
Base 16: **5F**

Base 2: **11111000**
Base 10: **-8**
Base 16: F8

Base 2: 10 1101
Base 10: **45**
Base 16: **2D**

Base 2: **11111101**
Base 10: -3
Base 16: **FD**

Bit Operators

& - Bitwise And

- 1 & 1 = 1
- 1 & 0 = 0
- 0 & 1 = 0
- 0 & 0 = 0

| - Bitwise Or

- 1 | 1 = 1
- 1 | 0 = 1
- 0 | 1 = 1
- 0 | 0 = 0

^ - Xor

- 1 ^ 1 = 0
- 1 ^ 0 = 1
- 0 ^ 1 = 1
- 1 ^ 1 = 0

<< - Left Shift

- 1 << 1 = b10
- 1 << 2 = b100
- 1 << 3 = b1000

>> - Right Shift

- 2 >> 1 = 1
- 2 >> 2 = 0
- 2 >> 3 = 0

Notes:

- 2 >> -1 = undefined!
- 2 << -1 = undefined!

Logical (Boolean) Operators

&& - Logical And

T && T = T

T && F = F

F && T = F

F && F = F

|| - Logical Or

T || T = T

T || F = T

F || T = T

F || F = F

! - Logical Not

!T = F

!F = T

Boolean logic tricks

- What is the binary representation of the smallest 2C 16-bit integer?
- How to get -1 in binary without using - sign?
- !!x is not x
- Bitmask: &(~0), &0, & 0xFF
- -1 + 1 = 0
- x ^ 0; x ^ -1
- Setting a bit x | (1 << 2);
- Clearing a bit x & ~(1 << 2);
- Flip a bit: x ^ (1 << 2);

Logic Simplification

Identity

- A & 1 = A
- A & 0 = 0
- A | 1 = 1
- A | 0 = A
- ~~ A = A

Associative

- A & (B & C) = (A & B) & C
- A | (B | C) = (A | B) | C

Distributive

- A & (B | C) = (A & B) | (A & C)
- A | (B & C) = (A | B) & (A | C)

More Identity

- A & A = A
- A | A = A
- A & ~A = 0
- A | ~A = 1

De Morgan’s Law

- ~(A & B) = ~A | ~B
- ~(A | B) = ~A & ~B

CMOS and Logic Gates

CMOS

PUN (Pull Up Network)

- Comments output to 1 (Vdd)
- pMOS Transistors

PDN (Pull Down Network)

- Connects output to 0 (Ground)
- nMOS Transistors

PUN and PDN should be complimentary (Series and Parallel Gates)

Design

Start with PDN (when boolean expression evaluates to zero - negate expression)

Negat PDN expression to create complimentary PUN

Logic Gates

AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

XOR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

NAND

A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

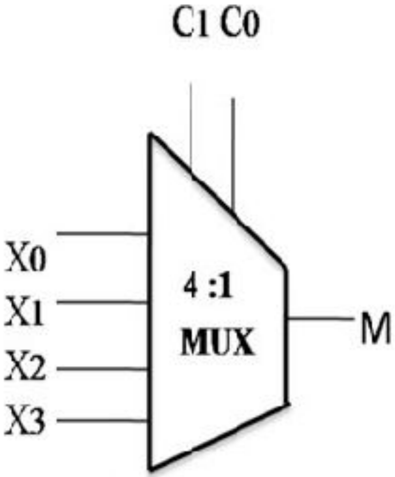
NOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

MUX



(a)

C1	C0	M
0	0	X0
0	1	X1
1	0	X2
1	1	X3

(b)

CMOS and Logic Gates Practice

Practice

Question 1 {25 pts}

Your job is to design a circuit that will take as input a 4-bit value and produces the output 1 if and only if the first two bits of the 4-bit value are the same as the last two bits. For example, if I = 1010 then the circuit should output 1. If I = 1001 then the circuit should output 0. In your diagram I_3 I_2, I_1 and I_0 should indicate the 4 bits of the input where I_3 is the MSB and I_0 is the LSB. Remember that we cannot grade what we cannot read so please make your diagrams as neat and clear as possible.

Part 1 {3 pts}: List all possible values for the input I that can result in the output being 1. You do **not** have to list any input values that result in the output being 0. Please circle your answer.

Practice

1111

1010

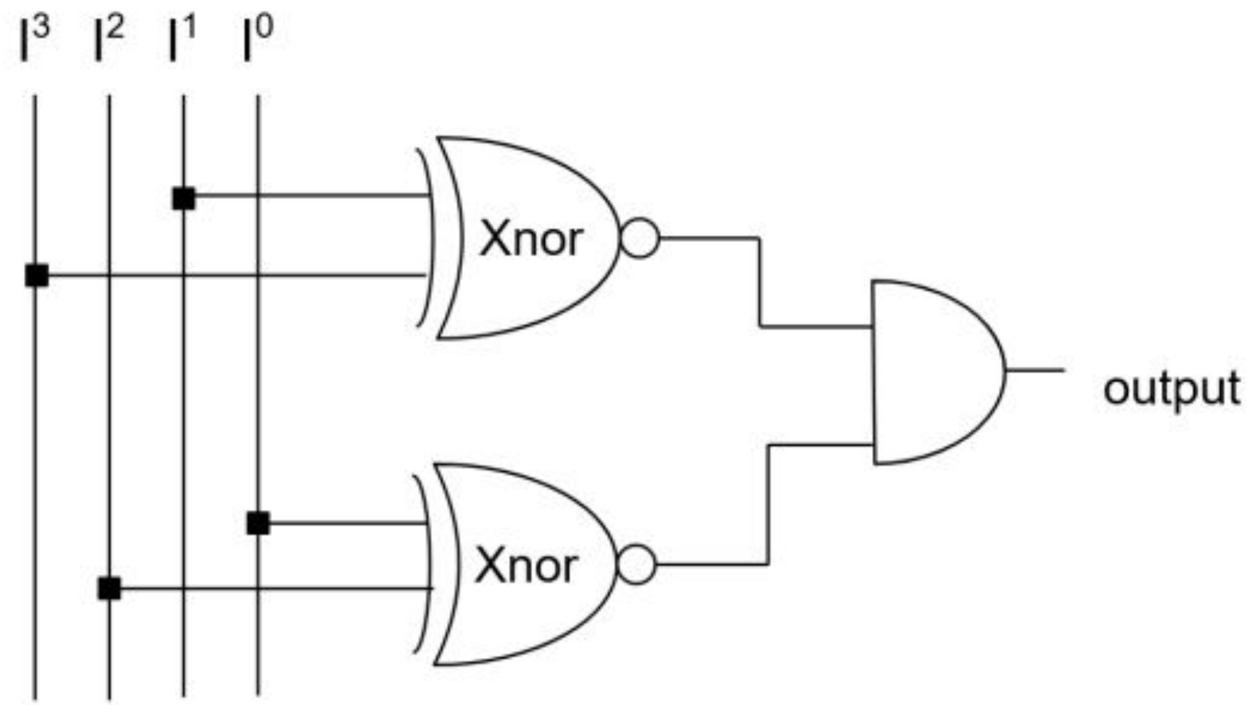
0101

0000

Practice

Part 3 {6 pts}: Design a gate-level **non-PLA** circuit that takes in the 4-bit input I and produces the correct output. Please label the inputs and outputs of your circuit clearly on your schematic.

Practice



Practice

Part 4 {10 pts}: Design a proper **CMOS** circuit which takes in all 4 bits of the input I and produces the output bit. You can assume that you also have access to negated versions of all of the input bits. Your solution must be a single CMOS circuit consisting of complementary pull up and pull down transistor networks. It should not involve cascading multiple CMOS circuits. Please label the inputs and outputs of your circuit clearly on your schematic.

Practice