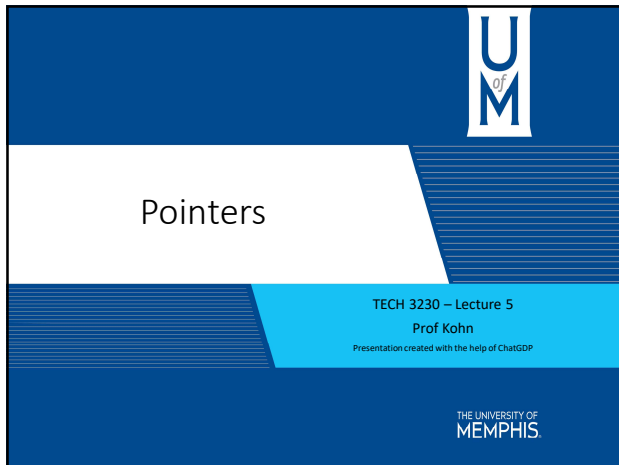
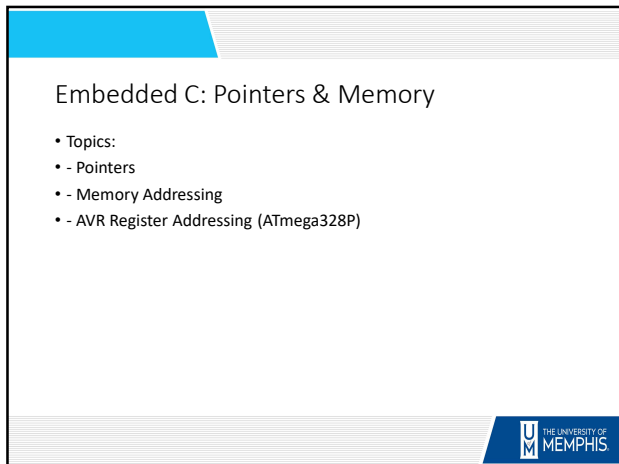
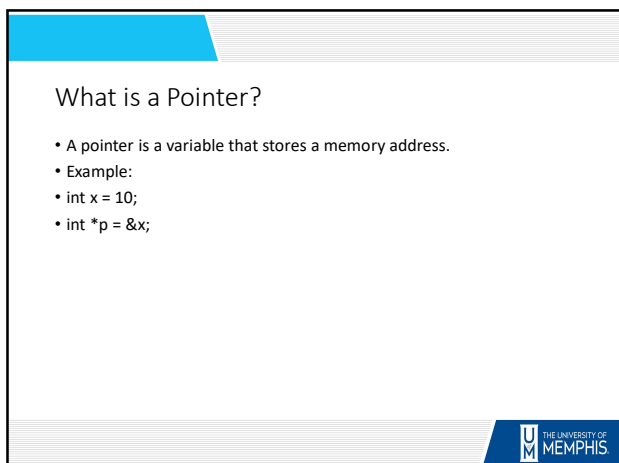




1



2



3

Pointer Basics

- Operators:
- `&` : address-of
- `*` : dereference
- Example:
- `*p = 20; // changes x`



4

Pointer Example

- `int x = 5;`
- `int *p = &x;`
- `printf("%d", *p); // prints 5`



5

Pointer Arithmetic

- Pointers can be incremented:
- `p++` moves to next memory location
- Based on data type size



6

Memory Addressing

- Memory is a collection of addresses.
- Each variable is stored in a specific location.



7

Example Memory Layout

- `int a = 10;`
- `int b = 20;`
- `&a` and `&b` give their memory addresses.



8

Why Memory Matters in Embedded

- - Limited RAM
- - Direct hardware access
- - Efficient code



9

ATmega328P Overview

- Microcontroller used in Arduino Uno
- Has registers mapped to memory addresses



10

Register Addressing

- Registers are special memory locations
- Used to control hardware



11

Example: DDRB Register

- DDRB controls pin direction
- 1 = output, 0 = input



12

Accessing Registers in C

- `#define DDRB (*(volatile unsigned char*)0x24)`
- `DDRB = 0xFF; // set all pins as output`



13

PORTB Example

- `#define PORTB (*(volatile unsigned char*)0x25)`
- `PORTB = 0xFF; // set pins HIGH`



14

Bit Manipulation

- `DDRB |= (1 << 2); // set bit 2`
- `DDRB &= ~(1 << 2); // clear bit 2`



15

Pointer to Register

- `volatile uint8_t *port = (uint8_t*)0x25;`
- `*port = 0xFF;`



16

Why volatile?

- Prevents compiler optimization
- Ensures hardware register is always read/written



17

Summary

- - Pointers store addresses
- - Memory is directly accessed
- - Registers are memory-mapped
- - Essential for embedded systems



18
