CprE 288 – Introduction to Embedded Systems
Exam 1 Review

Instructor: Dr. Phillip Jones

Dr. Zhao Zhang

Overview of Today's Lecture

- Announcements
- Exam 1 Review

Announcements

- Homework 5 is due next Thursday (turn in during class)
- Thursday, 9/27, Exam 1
  - Open book, open notes, calculator allowed
  - 1 hour and 15 minutes (starts from 9:30am and 2:30pm)

Announcements

Very important

Morning class: Do **NOT** disclose the exam questions

- Don't discuss the exam with anyone until the end of day

Afternoon class: Do **NOT** try getting any info about the exam

If you see any sign of cheating, report to the instructors/TAs

Any offender will be dismissed from the class immediately, with follow-on consequences from the department, college and/or university

EXAM FORMAT

Exam Format

- Focus on C programming, some questions may be specific to the AVR ATmega128 processor
- Open notes, no electronic devices except calculators
- Covers first 5 weeks of class
  - No questions about timers
  - No questions about interrupt
- 60 points total
  - 15% of your final grade

EXAM TOPICS

Exam Topics

Suggested preparation steps:

- Review the following slides. You should have a deep understanding of the content that appears on them
- If not, go back and review the lecture material on the given topic
- Run through the questions at the end of this PowerPoint

This set of slides are *not comprehensive*

- Review all lecture slides
- Review homework questions, try to re-do those questions

Exam Topic: Keywords

- char
- short
- int
- long
- float
- double

- enum
- struct
- union

- break
- case
- continue
- default
- do
- else
- for
- ~~goto~~
- if
- return
- switch
- while

- ~~auto~~
- ~~const~~
- extern
- ~~register~~
- signed
- static
- unsigned
- ~~volatile~~

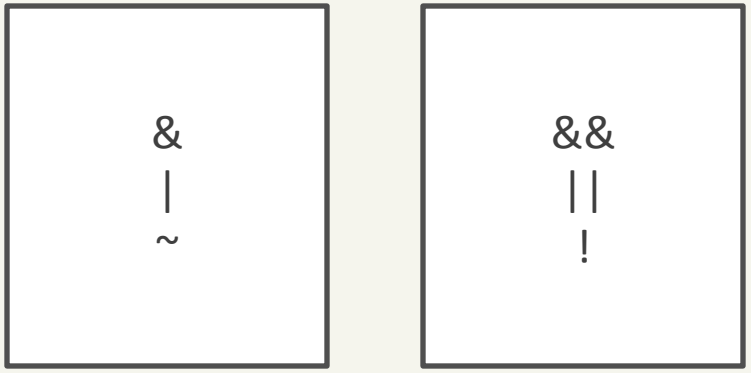
- ~~sizeof~~
- typedef
- void

Exam Topic: Syntax

- Could you write the following statements by hand?
  - Loops (for, while, do)
    - Write a **for loop** to sum elements of an array or count characters in a string
    - Do you know the syntax of a **do while loop**, **for loop**, and **while loop**?
  - typedef
    - Could you write a typedef definition
    - Do you know what it means when you see a variable type like **uint8_t**?
  - Switch statements
    - Do you know where the semi-colon and colons go in a switch/case statement?
    - Do you understand how the control flow can fall through a case?
  - Control flow
    - Do you understand the keywords **break** and **continue** and their use?

Know your Operators

- Do you know the difference between these two sets of operators?



Know your Operators

```
char a = 20, b = 10, r = 5;

// math operations
r = a + b;
r = a - b;
r = a * b;
r = a \ b;
r = a % b;

// bitwise operators
r = a & 3;
r = a | b;
r = a ^ 0xFF;
r = ~a;
r = a >> 3;
r = b << r;

// conditional
r = (r) ? a : b;
```

```
// boolean
r = a || b;
r = a && b;
r = !a;
r = a < 20;
r = b <= 15;
r = b > 10;
r = a >= b;

// post and prefix
a++;
++a;
b--;
--b;

// assignments
r = a = b = 42;
r += a;
r -= b;
```

Know your Operators

- Array access
- Pointers
  - Dereference
  - Address operator
- Access members of structs and unions

Pointers

- What are pointers
- Relationship between
  - pointers
  - array names
  - function names
- Pointer arithmetic

Know your Declarations

- Do these declarations make sense?

```
void main() {
    char x = 5, y = 10;
    char z;
    char array1[10];
    char array2[] = {1, 2, 3};
    char array3[5] = {1, 2, 3};
    char *str = "Hello!";

    int i = 7;
    int *ptr = &i;
    int **pp = &ptr;
    char *p;
}
```

Know your Declarations

- Do these declarations make sense?

```
struct House {
    unsigned long value;
    unsigned char baths;
    unsigned char bedrooms;
    unsigned char stories;
    unsigned long footage;
};

void main() {
    struct House my_home;
    struct House *bob_home = malloc(sizeof(House));

    my_home.baths = 1;
    my_home.value = 115000;
    bob_home->baths = 3;
    bob_home->value = 230000;
}
```

Know how to use Operator Precedence

- Can you use this table?

Precedence	Operator	Description	Associativity
1	`++ --`	Suffix/postfix increment and decrement	Left-to-right
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Element selection by reference	
	`->`	Element selection through pointer	
2	`++ --`	Prefix increment and decrement	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(type)`	Type cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of	
3	`* / %`	Multiplication, division, and modulus (remainder)	Left-to-right
4	`+ -`	Addition and subtraction	
5	`<< >>`	Bitwise left shift and right shift	
6	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
7	`== !=`	For relational = and ≠ respectively	
8	`&`		
9	`^`		
10	`|`		
11	`&&`		
12	`||`		
13	`?:`	Ternary conditional	Right-to-Left
14	`=`	Direct assignment	
	`+= -=`	Assignment by sum and difference	
	`*= /= %=`	Assignment by product, quotient, and remainder	
	`<<= >>=`	Assignment by bitwise left shift and right shift	
	`&= ^= |=`	Assignment by bitwise AND, XOR, and OR	
15	`,`	Comma	Left-to-right

QUESTIONS

Question 1

A. How many bytes are each of the following types (on the ATmega128)?
   - char, short, int, long, float, double

B. What range of values can be stored in an unsigned char?

C. What range of values can be stored in a signed char?

D. What is the value stored in **x** after this code runs?

```
int x, y, z;
x = y = z = 10;
```

Question 1 (answer)

Name	Number of Bytes sizeof()	Range
char	1	-128 to 127
signed char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
int (on ATmega 128)	2	-32,768 to 32,767
(pointer on ATmega 128)	2	Address Space
long	4	-2147483648 to 2147483647
signed long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
long long	8	-4294967295 to 4294967295
float	4	±1.175e-38 to ±3.402e38
double (on ATmega 128)	4	±1.175e-38 to ±3.402e38

Question 2

A. Analyze the following code:

```
char r = 0, s = 1, t = 2;
char *p1 = &s;
char *p2 = &t;
char **pp3 = &p1;

*p1 = 10;
**pp3 = 15;
*p2 = 30;
*pp3 = &r;
**pp3 = 5;
*p1 = 25;
```

Question 2 (answer)

A. Analyze the following code:

```
char r = 0, s = 1, t = 2;
char *p1 = &s;
char *p2 = &t;
char **pp3 = &p1;

*p1 = 10;          // s = 10
**pp3 = 15;        // s = 15
*p2 = 30;          // t = 30
*pp3 = &r;         // p1 = &r
**pp3 = 5;         // r = 5
*p1 = 25;          // r = 25
```

r	s	t
0	1	2
0	10	2
0	15	2
0	15	30
0	15	30
5	15	30
25	15	30

Question 3a

- When is the condition of the following **if** statement true?

```
if ((x = 3) || (x & 1)) {
   // do something
}
```

Question 3a (answer)

- When is the condition of the following **if** statement true?

```
if ((x = 3) || (x & 1)) {
   // do something
}
```

- The statement is always true. Know the difference between the assignment operator (=) and the equality operator (==).
  - The value on the left (x = 3) is always true, as the value of an assignment is the value that was assigned. This allows programmers to have compound assignments.

Question 3b

- When is the condition of the following **if** statement true?

```
if ((x == 3) || (x & 1)) {
   // do something
}
```

Question 3b (answer)

- When is the condition of the following **if** statement true?

```
if ((x == 3) || (x & 1)) {
   // do something
}
```

- The statement is true if x is either equal to 3 or bit 0 is set.

Question 4a

- When is the condition of the following **if** statement true?

```
if (x & 0x08 == 0x08) {
   // do something
}
```

Question 4a (answer)

- When is the condition of the following **if** statement true?

```
if (x & 0x08 == 0x08) {
   // do something
}
```

- The statement is true if bit 0 of x is 1. Operator precedence evaluates the == operator before the bitwise AND (&).

Question 4b

- When is the condition of the following **if** statement true?

```
if ((x & 0x08) == 0x08) {
   // do something
}
```

Question 4b (answer)

- When is the condition of the following **if** statement true?

```
if ((x & 0x08) == 0x08) {
   // do something
}
```

- The statement is true if bit3 of x is set.
  - x = 0b00001000; condition is TRUE
  - x = 0b01001110; condition is TRUE
  - x = 0b00101001; condition is TRUE
  - x = 0b00000000; condition is FALSE
  - x = 0b11100000; condition is FALSE

Question 6a

- What are the values of c1, c2, c3, and c4 after the following code executes?

```
char myarray[3] = {1, 2, 3};
char *ptr = myarray;

char c1 = *ptr++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

Question 6a (answer)

```
char myarray[3] = {1, 2, 3};
char *ptr = myarray;

char c1 = *ptr++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

- Postfix increment has higher precedence than dereference operator

- c1 is 1
- c2 is 2
- c3 is 1
- c4 is 2

Question 6b

- What are the values of c1, c2, c3, and c4 after the following code executes?

```
char myarray[3] = {1, 2, 3};
char *ptr = myarray;

char c1 = (*ptr)++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

Question 6b (answer)

```
char myarray[3] = {1, 2, 3};
char *ptr = myarray;

char c1 = (*ptr)++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

- c1 is 1
- c2 is 2
- c3 is 2
- c4 is 2

Question 7

```
/**
 * Returns the first index of occurrence of a given
 * character inside a string.  If not found, return -1.
 *
 * @param needle, the character to find in haystack
 * @param haystack, the string which is searched
 */
int find(char *haystack, char needle);
```

- Given this function signature, implement the function

```
void main() {
    find("hello world", 'c'); // returns -1
    find("hello world", 'h'); // returns 0
    find("hello world", 'o'); // returns 4
    find("hello world", 'w'); // returns 6
}
```

Question 7 (answer)

```
int find(char *haystack, char needle) {
    for (int i=0; haystack[i]; i++) {
        if (haystack[i] == needle)
            return i;
    }

    return -1;
}
```