10/26/2025

CS 211

QUIZ 4

(1) What is the result of this program:

```
#include <stdlib.h>
#include <stdio.h>

int* function( int* pointer ) {

    pointer = malloc ( sizeof(int) );
    pointer[0] = 1;
    return pointer;

}

int main() {

    int* array = NULL;

    function( array );

    printf ( "%d\n", array[0] );

    return 0;

}
```

← return is never captured

→ (* array) = * null

Error, dereferencing a null pointer

(2) What is the result of this program:

```
#include <stdlib.h>
#include <stdio.h>

int main() {

  int* array = calloc( 1, sizeof(char) );
  array[0] = 1;
  printf( "%d\n", array[0] );

  free ( array );

  return 0;

}
```

→ expects 4 bytes as it is int* but only has 1 byte allocated.

so even though I could fit in 1 byte it needs 4 allocated as it is an int*

Note that changing int* to char will make it work as expected.

Heap buffer overflow.

(3) What is the result of this program:

```
#include <stdlib.h>
#include <stdio.h>

int main() {

    int number = 100;
    int* pointer = &number;

    printf( "%d\n", *pointer );

    free(pointer);

    return 0;
}
```

← ptr to number

← prints 100

← can't free something if it wasn't malloced/calloced

will result in address sanitizer error.

(4) What is the result of this program:

```
#include <stdlib.h>
#include <stdio.h>

void function( int* pointer ) {

    printf( "%d\n", *pointer );
    free (pointer);

}

int main() {

  int* array = calloc( 1, sizeof(int) );
  array[0] = 1;

  function( array );
  free ( array );

  return 0;

}
```

← double free error

(5) What is the result of this program:

```
#include <stdlib.h>
#include <stdio.h>

int* function( int* pointer ) {

    pointer = malloc ( sizeof(int) );
    pointer[0] = 1;
    return pointer;

}

int main() {

    int* array = NULL;

    array =  function( array );

    printf ( "%d\n", array[0] );

    return 0;

}
```

Error. no free for malloc

memory leak.

→ will still print 1 however.

(6) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    unsigned char number = 128; // 1000_0000
    number = ~number;
    printf("number = %d\n", number);
}
```

1000_0000 = 128

~1000_0000 = 0111_1111 = 64+32+16+8+4+2+1 = 127

will print out number = 127.

(7) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    signed char number = 127; // 0111_1111
    number = ~number;
    printf("number = %d\n", number);
}
```

1000_0000 = -128

number = -128

(8) In both C and Java, hexadecimal literals are prefixed by "0x" and octal literals are prefixed by "0".

The latter syntax can be unexpected and tricky if you are not expecting the compiler to interpret your number as octal!

You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    printf("%d, %d\n", 0x10, 010 - 1);
}
```

16, 7

0x10 010-1

0001 0000 | 001 000-1

0 000 1

0000 0111

(9) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    unsigned short number = 255; // 0000_0000_1111_1111
    number++;
    printf("number = %d\n", number);
}
```

→ 4 bytes for possible numbers instead of 2 for negative and 2 for positive.

0000 0001 0000 0000

will print number = 256

10/26/2025

CS 211

QUIZ 4

(10) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    unsigned char number = 255; // 1111_1111
    number++;
    printf("number = %d\n", number);
}
```

unsigned char can only hold 2 bytes
100000000
truncates to 0
number = 0

(11) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    signed char number = 127; // 0b0111_1111
    number++;
    printf("number = %d\n", number);
}
```

can hold 2 bytes but has signed numbers
← max positive num
10000000 → -128
number = -128

(12) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    signed char number = -16; // 1111_0000
    number = number>>3;
    printf("number = %d\n", number);
}
```

right shift repeats the sign bit due to it being a signed char
right shift: 1111 1110 = -2
number = -2
-128
+126
-2

(13) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    signed char number = 1; // 0000_0001
    number = number<<7;
    printf("number = %d\n", number);
}
```

no repeating bits for left shift, it just pads 0s
1000 0000
-128
number = -128

(14) You compile and run this program on the iLab machines. What is printed to the command line?

```
#include <stdio.h>
void main() {
    char* sign_reps[4] = {
        [0] = "weird",
        [1] = "sign and magnitude",
        [2] = "ones' complement",
        [3] = "two's complement",
    };
    char* this_machine = sign_reps[-1&3];
    printf("This machine's sign representation is: %s.\n", this_machine);
}
```

1111 & 0011
0011
This machine's sign representation is: twos complement.

(15) You compile and run this program. What is printed to the command line?

```
#include <stdio.h>
void main() {
    printf("2^1 = %d\n", 2^1); // Notice that this is the bitwise XOR operator; it does not mean exponentiation
}
```

2^1 = 3
or 10 / 01 → 11