EC310
6-week Review
Rules of Engagement

- Teams selected by instructor
- Host will read the entire questions. **Only after**, a team may “buzz” by raise of hand
- A team must answer the question within 5 **seconds** after buzzing in (must have answer at hand)
- If the answer is incorrect, the team will lose its turn and another team may buzz in. No score will be deducted. No negative scores.
- Maximum score is 100. Once reached, that team will stand down for others to participate. Teams will earn all points scored at the end of game.
- When selecting a question, Teams must only select questions of different value, unless there are no others, but may be from different categories.
- All team members will participate and will answer questions
- Only one round - No Daily Doubles, Double Jeopardy or Final Jeopardy … and no partial credits!
## Jeopardy!

<table>
<thead>
<tr>
<th>Computer Architect &amp; Number Systems</th>
<th>C</th>
<th>Assembly and Memory</th>
<th>Pointers, Arrays &amp; Strings</th>
<th>Functions and Stack / Heap</th>
<th>Buffer Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 pts</td>
<td>10 pts</td>
<td>10 pts</td>
<td>10 pts</td>
<td>10 pts</td>
<td>10 pts</td>
</tr>
<tr>
<td>20 pts</td>
<td>20 pts</td>
<td>20 pts</td>
<td>20 pts</td>
<td>20 pts</td>
<td>20 pts</td>
</tr>
<tr>
<td>40 pts</td>
<td>40 pts</td>
<td>40 pts</td>
<td>40 pts</td>
<td>40 pts</td>
<td>40 pts</td>
</tr>
<tr>
<td>60 pts</td>
<td>60 pts</td>
<td>60 pts</td>
<td>60 pts</td>
<td>60 pts</td>
<td>60 pts</td>
</tr>
</tbody>
</table>
Properly define the items below.

**Hardware:** the electrical and mechanical components that make up a computer system

**Software:** the instructions that tell the computer how to do a specific job

**Main memory:** (aka: RAM) it stores the input data, instructions that will process the data, and results of calculations until they are moved to a permanent location. It is volatile.

**Secondary memory:** a more permanent memory for long-term storage of data before and after it is used. Ex. Hard drive, CD, DVD, flash memory (USB thumb drives)

**CPU:** Central Processing Unit. It is the “brain” of the computer. It executes the instructions that process data

**OS:** the most important software. Acts as an interface between the computer and the user.
What is the exact output of the C code below?

```c
#include <stdio.h>
{
    int pie;
    pie = 3.14;
    printf (pie);
}
```

Ans = 3
(pie is an integer)
Assembly and Memory  10 pts

How many bits are represented in an address?

4 bits per number, 8 numbers per address = 32 bits
Pointers, Arrays & Strings   10 pts

What is the address operator used to access the address of a variable?
True or False: The Heap grows from the top (smaller memory address) down (to larger memory addresses) making it grow towards the stack. In fact the stack and the heap compete for memory space and it is the programmer who must take care of the heap.

True
What feature of C makes a buffer overflow attack possible?

C compilers do not check for going beyond the bounds of an array!
Express the binary number $01011010$ as a decimal number.

Ans $= 64 + 16 + 8 + 2 = 90_{10}$
How many total bytes would be needed to store the following variables?

```c
int time_1, time_2, time_3;
float PRT = 9.5;
char mid_1, mid_2, mid_3;
int score;
```

- integers: \(4 \times 4 = 16\)
- floats: \(1 \times 4 = 4\)
- characters: \(3 \times 1 = 3\)

**TOTAL = 23 bytes**
Assembly and Memory  20 pts

- Name the processor registers and their purpose:

  **eip** (instruction pointer) – it holds the address of the next instruction the CPU intends to execute

  **esp** (stack pointer) – it holds the address of the top of the stack

  **ebp** (base pointer) – points to the first address right below the stack
What would be the output of the code below?

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char phrase[] = "Military Academy";
    strcpy (phrase, "Naval Academy");
    printf("%s\n", phrase);
}
```

Naval Academy
Functions and Stack / Heap  20 pts

- What elements are stored in memory and in what order are they stacked during a function call from main?

  Function’s Variables
  Saved value of prior ebp
  Return Address
  Function’s Arguments
  Main’s Variables
Buffer Overflow  20 pts

For the following program invocation:

midshipman@EC310 ~ $ ./a.out one 8 mate

a) What is the value of `argc`?
b) What is the value of `argv[1]`?
c) Is it possible to write these as the first instructions in `main`? Why?

```c
int i;
i = argv[2];
```

a) 4  
b) one  
c) No, `argv[2]` is of type `string`
Convert the decimal number 300 to hexadecimal

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$16^2=256$</td>
<td>$16^1=16$</td>
<td>$16^0=1$</td>
</tr>
</tbody>
</table>

$300 \div 256 = 1.xxx \quad 1$

$300 - 256 = 44$

$44 \div 16 = 2.xxx \quad 2$

$44 - 32 = 12 \quad C$

$Ans = 0x12C$
Consider the code below. What is the exact output if the user enters 4 when prompted?

```c
#include<stdio.h>
int main()
{
    int number , counter;
    printf( "Enter a number: " ) ;
    scanf( "%d" , &number );

    if( number != 5 )
        printf( "Is %d your favorite number?\n", number );
    else
        printf( "Navy\n" );

    for(counter = number ; counter < 8 ; counter = counter * 2)
        printf( "What’s for lunch?\n" );
}
```

Is 4 your favorite number?
What’s for lunch?
Consider the screen capture.

a) What is the next assembly instruction to be performed?

b) At what address is this instruction stored?

c) What would be the value of eip?

a) mov DWORD PTR [esp], 0x8048474
b) 0x8048384
c) 0x8048384
What is the output of the program below?

#include <stdio.h>
int main()
{
    int salary[4] = {1000, 1500, 2000};
    int j;
    for(j = 0 ; j <= 3; j = j + 1)
    {
        printf("Salary %d is %d \n", j+1, salary[j]);
    }
}

By initializing only the first part of the array, the remaining elements are initialized to zero.

Salary 1 is 1000
Salary 2 is 1500
Salary 3 is 2000
Salary 4 is 0
Consider the code below where a break has been set at line 10. Fill the missing items in the stack?

```c
void test_function(int a, int b, int c, int d, int e)
{
    int flag;
    char buffer[10];

    flag = 256;
    buffer[0] = 'U';
    buffer[1] = 'S';
    buffer[2] = 'A';
}

int main()
{
    test_function(5,6,7,8,9);
}
```

Hex    ASCII
0x41    A
0x53    S
0x55    U
Consider the code below. What is the *minimum* amount of characters needed in the string `alpha_code` to change the value of `a` in the stack?

```c
float happy_times(int x, float y) {
    char alpha_code[7];
    float sum;
    sum = x + y;
    Return sum;
}

int main() {
    int a = 77;
    float b = 3.14159;
    a = happy_times(a,b) + 1;
    exit(1);
}
```

- `alpha_code` = 7
- `saved ebp` = 4
- `Ret Add` = 4
- `copy of x` = 4
- `copy of y` = 4
- `b` = 4
- “change value of a” = change 1 byte

...so TOTAL = 7+4+4+4+4+4+4 = 27
What is the hexadecimal number that results from the calculation: $0x1A6 + 0xD97$

Solution:
Starting from the right…
6 + 7 = 13 = 0xD
A + 9 = 19 = 16 + 3 = 0x13 (the 1 is carried over)
1 + D = 14 + 1(carried over) = 15 = 0xF
So…

Ans = $0xF3D$
C 60 pts

Complete the code below to return the factorial of a number (hint: use for loop)

```c
#include<stdio.h>
int main ( )
{
    int number, counter;
    printf("Enter a number and I will return its factorial:");
    scanf( "%d" , &number );
    //enter 2 lines
    //of code here
    printf("The factorial is: %d \n", number );

    for (counter = number; counter > 1; counter = counter-1)
        number = number * (counter-1);

    for (counter = 1; counter < number; counter = counter+1)
        number = number * counter;
}```
Consider the memory stack allocation below. What would be displayed by these commands?

```
x/s 0x08048385
x/xw 0x08048384
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>08048384</td>
<td>0x55</td>
<td>U</td>
</tr>
<tr>
<td>08048385</td>
<td>0x53</td>
<td>S</td>
</tr>
<tr>
<td>08048386</td>
<td>0x4e</td>
<td>N</td>
</tr>
<tr>
<td>08048387</td>
<td>0x41</td>
<td>A</td>
</tr>
<tr>
<td>08048388</td>
<td>0x00</td>
<td>null</td>
</tr>
</tbody>
</table>

SNA

0x414e5355
Consider the code and the stack below. What would be the output of the program?
(assume the stack is contiguous without padding between addresses)

```
#include<stdio.h>
int main()
{
    int a = 11;
    int *a_ptr;
    a_ptr = &a;
    printf("The value of a is %d and the address is %x \n", a, &a);
    printf("The value of a_ptr is %x and the address is %x \n\n", a_ptr, &a_ptr);
}
```

The value of a is 11 and the address is bffff850
The value of a_ptr is bffff850 and the address if bffff84c
Consider the code below. Give the different variable values at specific break points.

```c
#include <stdio.h>
int AbsVal(int number) {
    int AV;
    if(number >= 0) {
        AV = number;
    } else {
        AV = -1*number;
    }
    return AV;
}

int main() {
    int x, y;
    x = -1;
    y = AbsVal(x);
    printf("The absolute value of the integer is %d\n", y);
}
```

<table>
<thead>
<tr>
<th>Breakpoint</th>
<th>x</th>
<th>y</th>
<th>number</th>
<th>AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 18</td>
<td>garbage</td>
<td>garbage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 19</td>
<td>-1</td>
<td>garbage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 5</td>
<td>-1</td>
<td>garbage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 13</td>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Line 20</td>
<td>-1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Consider the code below. A break was set at line 15, at which time the command `i r ebp` returned the value `0xbfffff800`. The code was allowed to continue until it prompted its message. Would a segmentation fault occur if the user inadvertently enters 14 characters? Explain.

```c
#include<stdio.h>
void echo_string()
{
    int count;
    char entered_string[10];
    printf("Enter a string: ");
    scanf("%s", entered_string);
    for(count=0; count < 10; count=count+1)
    {
        printf("%s\n",entered_string);
    }
}
int main()
{
    echo_string();
}
```

No. Although 14 characters would cause a byte to overwrite saved_ebp, the null would only replace byte 0x00, so the value does not change and therefore there is no access to memory outside that that was allotted and thus no SegFault.