CSCI-UA.0201
Computer System Organization
Homework Assignment 1

1. What will happen if, in a C program, you assign a value to an array element whose index exceeds the size of array (e.g. A[15] = 8 while A is an array of 14 elements only)? You must specify what will be the compiler reaction, and, if the compilation is successful, what will happen during execution.

2. In C, if you pass an array as an argument to a function, what actually gets passed?

3. What will be the output of the following program? Explain how you reached your solution.

```c
#include<stdio.h>

int main()
{
    int a[5] = {5, 1, 15, 20, 25};
    int x, y, z;
    x = ++a[1];
    y = a[1]++;
    z = a[x++];
    printf("%d, %d, %d", x, y, z);
    return 0;
}
```

4. We have seen in class that a[i] is equivalent to *(a+i). What is the equivalent of a[i][j] (assuming the 2D array has been allocated using a malloc of an array of pointers followed by a loop that mallocs each row, like what we say in slide 11, lecture “Machine Level V”)?

5. Is there anything wrong with the following declaration? If so, what is it?

```c
struct emp
{
    int ecode;
    struct emp *e;
};
```
6. Is there anything wrong with the following code? If so, what is it?
struct emp
{
    int ecode;
    struct emp e;
};

7. Assume we are executing the following program on a 64-bit machine:

   a) What will be the output?

   b) If we change p’s declaration to be: double * p; and use the type casting with malloc to be (double *), what will be the output?

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

int main()
{
    int *p;
    p = (int *)malloc(20);
    printf("%d\n", sizeof(p));
    free(p);
    return 0;
}
```
8.
Assume we are running code on a 6-bit machine using two’s complement arithmetic for signed integers. A “short” integer is encoded using 3 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

```c
short sy = -3;
int y = sy;
int x = -17;
unsigned ux = x;
```

Note: You need not fill in entries marked with “-”.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Decimal Representation</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 0010</td>
<td></td>
</tr>
<tr>
<td>ux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-TMin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-TMin and Tmax are the minimum and maximum two’s complement numbers. Note: You just need to fill the entries in “Decimal presentation” and “Binary presentation” columns.
6.

```c
void foo(int x)
{
    printf("address of x is \%p\n", &x);
    bar(x-1);
    return;
}

void bar(int y)
{
    printf("address of y is \%p\n", &y);
    return;
}

void foo2()
{
    bar2();
}

void bar2()
{
    char buf[10];
    gets(buf);
}
```

A. Suppose we invoke the function `foo` many times. What is the relationship of the address of x and that of y in resulting printf statements?
(a). address of x is always less than address of y.
(b). address of x is always greater than address of y.
(c). address of x is sometimes less than address of y and sometimes greater than address of y.

B. The `gets(s)` function reads a line from stdin into the buffer pointed to by s and does not check for buffer overrun. Suppose we invoke the function `foo2` and the user types in some line longer than 10 characters with the intent of exploiting the buffer overrun to execute malicious code by overwriting a return address. Which function would that return address have been pointing too if the attack had not occurred? (That is, which function would we have returned to).