Question 1

Consider the following declarations and assignments:

```c
int a, *b, *c, d[4];
for (a = 0; a < 4 ; a++) {
    d[a] = 1 + a;
}
b = d;
c = &a;
a = b[3];
```

Describe the **type** and **value** of each of the expressions below. The **type** should be one of: `int`, `int *` (int pointer), or `int []` (int array). For the **value**, if the expression is an address, describe what it is the address of. If an expression is invalid, write “Illegal Expression”.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. a</td>
<td>-----------------</td>
</tr>
<tr>
<td>2. b</td>
<td>-----------------</td>
</tr>
<tr>
<td>3. *b</td>
<td>-----------------</td>
</tr>
<tr>
<td>4. c</td>
<td>-----------------</td>
</tr>
<tr>
<td>5. d</td>
<td>-----------------</td>
</tr>
<tr>
<td>6. &amp;d[1]</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
Question 2

Trace through the following C code, and draw the contents of memory (heap and stack) at the point indicated, and show the output produced by a complete run of the program. (Assume stdio.h and stdlib.h have been included, and that malloc always succeeds.) Remember that the stack grows from the bottom and the heap grows from the top.

```c
int *func(int *a, int *b, int s);

int main (void) {
    int *arr = NULL, x = 4, y = 3, i;

    arr = func(&x, &y, 5);
    printf("x = %d  y = %d\n", x, y);
    if (arr != NULL) {
        for (i = 0; i < 5; i++) {
            printf("arr[%d] = %d\n", i, arr[i]);
        }
    }
    free(arr);
    return 0;
}

int *func(int *a, int *b, int s) {
    int *tmp, i;

    tmp = malloc(sizeof(int) * s);
    if (tmp != NULL) {
        for (i = 0; i < s; i++) {
            tmp[i] = i + *b;
        }
        *a = tmp[2];
        *b = 8;
    }
    // DRAW MEMORY WHEN YOU GET HERE
    return tmp;
}
```

OUTPUT
------