CS 240 **There was more space on original quiz** Spring 2010
Quiz 2 P. O'Neil

NAME_____

(1) (10 pts) Print the output values resulting from the following code fragment. **Careful, at least one of these statements has an effect on a later statement**. **SHOW WORK!!**

```
int x = 5, y = 2, r[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}, *ip;

ip = &r[1] + 2;

x = *ip;

printf ("%d, %d" , x, *(ip+1));        ANSWER:

printf ("%d", ip[2]++);                ANSWER:

y = *(&r[1]+4);

ip -= 2;

printf ("%d, %d", *ip, y);             ANSWER:
```

(2) (10 pts-Originally said 15) Write a function memdup with the following functional prototype:

```
void *memdup(void *obj, int size);
```

The argument obj is a pointer to some object, probably a struct or an array, and the argument size is the size in bytes of the object. The function memdup should malloc space for a new copy of the object, and copy the object pointed to by obj to the newly malloced space, returning the pointer to the new copy. Note that memdup() should return NULL if malloc returns an error. You need the library function memcpy() to copy the object; strcpy() won't work!

CS240 Quiz 2 Solutions, Spring 2010

(1) int x = 5, y = 2, r[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}, *ip;
0 1 2 3 4 5 6 7 8 9

ip = &r[1] + 2; Pointer &r[1] points to 4, +2 points to 8.

x = *ip; Thus x = 8 / and ip+1 points to 10, so *(ip+1) is 10

printf ("%d, %d" , x, *(ip+1)); ANSWER: 8, 10

/ ip[2] is 12, gets incremented after the fact (set to 13 above)

printf ("%d", ip[2]++); ANSWER: 12

y = *(&r[1]+4); y = (basically) r[5] = 13 (now)

ip -= 2; ip now points to 4
printf ("%d, %d", *ip, y); ANSWER: 4, 13

(2)
```
void memdup(void *obj, int size)        {
    void * temp;
    if((temp = malloc(size)) != NULL)
        memcopy(temp, obj, size);
    return temp;
}
```
Most common errors: act like obj is some known struct, write struct obj * temp; as a declaration. We DON'T KNOW the type obj points to, it's just a void *.

For this reason, when we malloc we MUST use the size argument passed, we can't say sizeof(*obj) or anything like that; we DON'T KNOW THE TYPE of the thing pointed to by obj.
Not using size in malloc was an automatic 3.5 points off (the whole problem was 10 points.

The next most common error was to use strcpy (which I told you not to use) or to use memcpy with two arguments, meaning you didn't look it up in K&R. Both of these result in at least 5 points off.