# Midterm Exam Solutions

## 1. Short Answer

### Part 1: Two's Complement

```
10101100
```

### Part 2: Errors vs. Leaks

The core cause of a memory error is accessing (reading or writing) memory that does not belong to you. For instance, if you create a string that is not null-terminated and call strlen() on it, this causes a memory error because it will cause strlen to search through memory that doesn't belong to that string in search of a null terminator.

The core cause of a memory leak is not freeing memory you have allocated on the heap. For instance, if you call malloc() or strdup(), you must free the return value of that function when you are done with it.

### Part 3: Memory Errors

The first memory issue is the call to realloc does not correctly specify the new larger size; it uses the number of elements instead of the number of bytes. The realloc size should multiply the specified value by sizeof(int).

The second memory issue is a double-free at the end of the code; in both cases, we should only free new_arr, not arr. If realloc resizes in place, then new_arr and arr will point to the same memory, so we should free only once. If realloc moves the memory, then arr will point to freed memory.

## 2. C Strings

*Sample Solution*

```
char *remove_delimiters(const char *str, const char *delimiters) {
    size_t alloc_size = INITIAL_SIZE;
    char *new_str = malloc(alloc_size);

    // strcat assumes the string is null terminated.
    // we must also start with an empty string if there are no tokens.
    new_str[0] = '\0';

    char buf[INITIAL_SIZE];
    while (scan_token(&str, delimiters, buf, sizeof(buf))) {
        // resize if we are out of space
        if (strlen(buf) + strlen(new_str) + 1 > alloc_size) {
            new_str = realloc(new_str, 2 * alloc_size);
            alloc_size *= 2;
        }

        strcat(new_str, buf);
    }

    return new_str;
}
```

## 3. Extract Min

**Part 1: `extract_min`**

*Sample Solution*

```
void extract_min(void *base, void *dest, size_t nelems, size_t elem_size_bytes,
                 int (*cmp_fn)(const void *, const void *)) {

    // Find minimum element - if multiple, return any one
    void *min_elem = base;
    for (int i = 1; i < nelems; i++) {
        void *ith = (char *)base + i * elem_size_bytes;
        if (cmp_fn(ith, min_elem) < 0) {
            min_elem = ith;
        }
    }

    // Copy minimum element to destination
    memcpy(dest, min_elem, elem_size_bytes);

    // Remove minimum element
    void *end = (char *)base + nelems * elem_size_bytes;
    memmove(min_elem, (char *)min_elem + elem_size_bytes,
            (char *)end - ((char *)min_elem + elem_size_bytes));
}
```

**Part 2: `cmp_strings_asc`**

*Sample Solution*

```
int cmp_strings_asc(const void *a, const void *b) {
    return strlen(*(const char **)a) - strlen(*(const char **)b);
}
```

**Part 3: calling `extract_min`**

```
// write parameter 1 below
strs        or &strs[0]
// write parameter 2 below
&min
// write parameter 3 below
nelems
// write parameter 4 below
sizeof(strs[0])   or sizeof(char *)       or sizeof(*strs)        or 8
// write parameter 5 below
cmp_strings_asc
```

## 4. Bits and Bytes

**Part 1: `get_bit_range`**

```
// write expression 1 below
bits >> rightmost_index
// write expression 2 below
(1L << (leftmost_index – rightmost_index + 1)) - 1
// write expression 3 below
shifted & mask
```

**Part 2: mask**

```
// write expression 1 below
(1L << (leftmost_index + 1)) - 1
// write expression 2 below
(1L << rightmost_index) – 1
```