Midterm Exam Solutions

1. Short Answer

Part 1: -107 as binary

```
10010101
```

Part 2: Error ID 1

The main issue is the returned string's characters are stored on the stack because it is a local variable character array. This means the function will return a pointer to stack memory that goes away when the truncate function finishes. In order to return a string from a function, it must be heap-allocated.

Note: for this problem, we gave credit if you identified a less-severe, but also important, memory issue with strncpy copying in too much if length > strlen(str).

Part 3: Error ID 2

The issue is truncate modifies its own copy of dest by setting it equal to heap-allocated memory, but this does not modify the char * passed in as a parameter because we make a copy of the provided address when calling the function. In order to modify a passed-in parameter itself, we must pass in the address of the value we wish to modify, and dereference it in the function to modify it.

2. Error-Correcting Codes

Expression A

```
message >> (3 * i)
```

Expression B

```
// blocks encoding 0 are powers of 2, except for 000, but this works there too!
block & (block – 1)
```

Expression C

```
// for this problem it is technically ok without L, since i is always < 32
| (1L << i)
```

3. Find and Replace

Sample Solution

```
char *find_and_replace(const char *str, const char *find, const char *replace) {
      char buf[MAX_STR_LEN] = "";

      while (true) {
            char *next = strstr(str, find);
            if (!next) break;

            // Copy up to this token
            strncat(buf, str, next - str);

            // Copy in value
            strcat(buf, replace);

            str = next + strlen(find);
      }

      // copy in the rest of the string, and return a heap copy
      strcat(buf, str);
      return strdup(buf);
}
```

4. Get Resized

Part 1: `get_resized`

Sample Solution

```
void *get_resized(void *base, size_t nelems,
                  size_t elem_size_bytes, size_t *p_nelems,
                  size_t (*times_fn)(void *)) {
    int count = 0;
    void *arr = NULL;

    for (int i = 0; i < nelems; i++) {
        void *elem = (char *)base + i * elem_size_bytes;
        size_t copies = times_fn(elem);
        arr = realloc(arr, (count + copies) * elem_size_bytes);
        for (size_t j = 0; j < copies; j++) {
            memcpy((char *)arr + count * elem_size_bytes, elem, elem_size_bytes);
            count++;
        }
    }

    *p_nelems = count;
    return arr;

}
```

Part 2: `times_int`

```
size_t times_int(void *ptr) {
    return *(int *)ptr;
}
```

Part 3: `times_string_length`

```
size_t times_string_length(void *ptr) {
    return strlen(*(char **)ptr);
}
```