

1. [45 pts.]

```
Student **read_students(void) {
    Student **p;
    float f;
    int y, i= 0;
    char n[MAX_NAME + 1];

    p= malloc(sizeof(Student *)); /* allocating initial array of pointers */
    if (p == NULL) {
        printf("Bad heap!\n");
        exit(-1);
    }

    while (scanf("%s %f %d", n, &f, &y) != EOF) { /* reading until EOF */

        p[i]= malloc(sizeof(Student)); /* allocating a Student for each line */
        if (p[i] == NULL) {
            printf("Bad heap!\n");
            exit(-1);
        }
        p[i]->name= malloc(strlen(n) + 1); /* allocating student's name */
        p[i]->age= malloc(sizeof(int)); /* allocating student's age */
        if (p[i]->name == NULL || p[i]->age == NULL) {
            printf("Bad heap!\n");
            exit(-1);
        }

        strcpy(p[i]->name, n); /* storing student's name */
        p[i]->shoe_size= f; /* storing student's shoe size */
        *(p[i]->age)= y; /* storing student's age */

        i++; /* keeping track of number of lines read */

        p= realloc(p, (i + 1) * sizeof(Student *)); /* making array larger */
        if (p == NULL) {
            printf("Bad heap!\n");
            exit(-1);
        }
    }

    p[i]= NULL; /* setting last element to NULL */

    return p; /* final return value */
}
```

```
void destroy_students(Student **s) {
    int i= 0;
    if (s == NULL || *s == NULL)
        return;
    while (s[i] != NULL) {
        free(s[i]->name);
        free(s[i]->age);
        free(s[i]);
        i++;
    }
    free(s);
}
```

2. [20 pts.]

a. `#define XOR(x, y) (((x) || (y)) && !((x) && (y)))`

Parentheses are required around the conjunct with `||`, since `||` has lower precedence than `&&`.

b. If `ch` happens to be a `char` variable with value between that of `'a'` and `'z'` (inclusive), or an `int` variable with value between whatever the ASCII values of `'a'` and `'z'` are, then it will print the corresponding character (e.g., if `ch` is a `char` variable with `'a'` then `a` will be printed, etc.). If `ch` has any other value then memory outside of the string literal will be referenced, leading to the unpredictable results (e.g., a segmentation fault).

c. The exponent value tells us how far to the right to move the binary point in the mantissa. Recall that the exponent value is biased by adding 127 to it. The value of the exponent field here, `1000 0010`, is 130.  $130 - 127 = 3$ , so the actual exponent is 3. Therefore we place a binary point three places to the right of the mantissa, producing `011.1010 0000 0000 0000 0000`.

Recall from class that the mantissa has an implied leading 1 which is dropped from the number in memory, so the actual mantissa value, adding that leading 1 back, is `1011.1010 0000 0000 0000 0000`.

The part to the left of the binary point, `1011`, has decimal value 11 ( $2^3 + 2^1 + 2^0 = 8 + 2 + 1 = 11$ ).

The part to the right of the binary point, `101` (followed by a whole lotta zeros), has decimal value

$$.625 \left( \frac{1}{2^1} + \frac{1}{2^3} = \frac{1}{2} + \frac{1}{8} = .5 + .125 = .625 \right).$$

Oh, forgot to say that the number is positive because the sign bit is 0. That means the final value in decimal is 11.625.

d. The child process inherits lots of attributes from the parent. Some from the lecture slides:

- program text (code)
- process credentials (user and group ID - UIDs and GIDs in UNIX terminology)
- environment
- runtime stack
- memory contents
- open file descriptors— `FILE *'`s from `fopen()`
- signal handling settings
- nice value (to determine process priority for scheduling)
- current working directory (set with `cd`, viewed with `pwd`)
- root directory
- controlling terminal (the program that controls `stdin`, `stdout`, `stderr` for the process, usually a shell)

3. [35 pts.]

```
irmovl    array, %eax    # load address of array into register
rmmovl    %eax, p        # and store into p
mrmovl    n, %ebx        # load n
irmovl    $4, %ecx
multl     %ecx, %ebx     # multiply n by 4 since an int is 4 bytes
addl      %eax, %ebx     # compute address into array
mrmovl    (%ebx), %edx   # get p[n]
wrint     %edx           # print p[n]
addl      %ecx, %eax     # finally, increment p
rmmovl    %eax, p        # and store into p
```