

1. [45 pts.] Consider the following structure type:

```
typedef struct {
    char *name;
    float shoe_size;
    int *age;
} Student;
```

Write the two functions described below:

```
Student **read_students(void);
void destroy_students(Student **s);
```

- The function `read_students()` should read data about students and return a pointer to a dynamically-allocated array storing the data read:
 - Your function should read until the end of the input is seen, where each input line contains the data for one student, consisting of a name (a single word which will always be less than 21 characters), a shoe size (a float), and an age (an integer), each separated by a single space.
 - Recall that the `scanf()` function will read a string using the `%s` format specifier, and `scanf()` will return EOF if the end of the input is detected. Therefore a loop like the following will read the input, assuming `name`, `foot_magnitude`, and `years_old` are declared appropriately.

```
while (scanf("%s %f %d", name, &foot_magnitude, &years_old) != EOF) {
    /* process the line of input which was read */
}
```
 - You may assume that the input will always be valid and of the form described, and when your function is called there will be one or more lines of input to be read.
 - Although you may assume there will be at least one line of input, other than that you have **no idea** how many lines of input there will be– there could be any number (but you can assume all the data will fit into memory).
 - Your function must return a pointer to an array that has exactly **one element more** than the number of lines in the input. Each element should be a **pointer** to a `Student` structure. The last element of the array must contain `NULL`, so the caller can tell where the end of the data is. As it works, your function may create arrays of whatever size you want, but when it leaves, it must return an array of **exactly the size described**, namely with one more element than the number of lines in the input.
 - Each student's name must be stored in a dynamically-allocated array (that the `name` field in a `Student` structure will point to) that is exactly the right size needed to store that name. The student's age will also be stored in a dynamically-allocated `int`, for no real good reason except that we want it that way.
 - To make things simpler you may assume that all memory allocations will always be successful.
 - We don't care how efficient (or inefficient) your function is.
 - Note that the function has to deal with a gazillion pointers – it must create a dynamically-allocated **array of pointers**, where each pointer in the array points to a **dynamically-allocated** `Student` structure. Each `Student` structure contains two dynamically-allocated fields (a student's name and age).

- The effect of the function `destroy_students()` is much easier to describe – it must release all the memory that its parameter `s` points to. If the parameter `s` is a `NULL` pointer, it should return without doing anything. Otherwise it may assume that the pointer points to a valid dynamically-allocated array that was created by an earlier call to `read_students()`, so it's of the format described above.

Comments are unnecessary, but your functions must be written *neatly*, with good style and formatting.

2. [20 pts.] **Briefly** answer the following short-answer questions.

- a. C has the logical inclusive-or operator `||`, but there is no logical exclusive-or operator. Write a complete macro `XOR(x, y)` that will return the logical exclusive or of its two parameters. (Recall the exclusive-or of x and y is true when either x or y is true, but not both of them, where we want the logical, not bitwise or.) A user should be able to invoke the macro as in the examples below. Note that the macro's arguments should be able to be arbitrary expressions, as in the second example.

```
if (a > b && XOR(c, d) == 1)          if (XOR(w && x, !y || z))
...                                     ...
```

- b. **Briefly** describe what results the following valid C statement would produce, based upon the value of the variable `ch`:

```
printf("%c", "abcdefghijklmnopqrstuvwxy"[ch - 'a']);
```

- c. Assuming that floats are stored in IEEE 754 format, below are the three parts of a single-precision (32-bit) number in binary. Give the value of the number in decimal. Show your work in order to be able to receive partial credit if applicable.

sign bit: 0 exponent: 1000 0010 mantissa: 0111 0100 0000 0000 0000 000

The decimal value of the number is: _____

- d. Name three attributes of the parent process that a child process inherits after a `fork()` system call.

