**CIS 308 Exam 1 (50 points)**

**Fall 2022**

**Name: \_\_\_\_\_\_\_\_\_\_\_Solution\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Lab section (Tuesday or Thursday): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**This test is closed-notes and closed-computers.**

There are 5 problems. The exam is printed on both sides of the paper – check each side to ensure you don’t miss any of the problems.

Score: \_\_\_\_\_\_\_\_\_\_\_\_

1. (12 pts) Suppose that we have the following variable declarations:

int \*first;

int second[6];

Further suppose that we have dynamically given *first* space for 8 elements (which have been initialized to {0,1,-1,2,-2,3,-3,4}). Suppose that *second* has the values {4,5,7,10,14,19}. State what is printed by each of the following statements (being as specific as you can), assuming that they are executed in order.

1. printf(“%d\n”, \*first);

0

1. printf(“%d\n”, \*first + 1);

1

1. printf(“%d\n”, second + 1);

memory address of the 5 (second[1]’s memory address)

1. printf(“%d\n”, &second);

memory address of second (pointer to a pointer)

1. first = \*(first + 3);

first--;

printf(“%d\n”, \*first);

prints contents of memory location 1

1. first = second + 1;

first+=2;

printf(“%d\n”, first[1]);

prints 14

1. (7 pts) Consider the following questions about arrays and strings.
   1. (1 pt) Declare a constant-sized array called *letters* that will hold 10 characters.

char letters[10];

* 1. (2 pts) Print a prompt to ask the user for their name and read the result into the *letters* variable. Do NOT assume the user will type fewer than 10 characters. If the user enters a name longer than that, only read in the first 9 letters.

printf(“Enter name: “);

fgets(name, 10, stdin);

* 1. (2 pts) Suppose the user entered the name “Alice” in step (b), and that the next lines in the program (after the user input) were:

letters[3] = ‘\0’;

printf(“%s\n”, letters);

What would print? Why?

Ali

(it would think the end of the string was at position 3)

* 1. (2 pts) Explain what would happen if the following statement came after part (c) in your program:

letters[10] = ‘\0’;

attempts to put a ‘\0’ at the position just past the end of the array (might crash, but more likely will overwrite some other spot in memory – possibly changing a variable in our program if it was stored at that memory address)

1. (12 pts) Write the function ***reverse*** that takes a string (character pointer) as an argument and updates that string to be its own reverse. (Note that this function should have a void return type.) For example, if the argument was originally “exam”, then it should be updated to be “maxe” when the function ends. You should not call any string library functions except *strlen*. **Use ONLY pointer notation – you should not use [] array brackets.**

Common errors:

using [] instead of pointer notation

undoing the reverse

if you used malloc, you needed to use free

void reverse(char\* word) {

//blah blah reverse

word = temp; //won’t change original string

}

Potential solution:

void reverse(char\* word) {

char\* start = word;

char\* end = word+strlen(word)-1;

while (start < end) {

char temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

}

1. (8 pts) Write the **call-by reference** C function ***absval*** that takes the address of a double as a parameter. This function should update the double to be the absolute value of its original value. (Do not call any C math library functions.) For example, if the number -6.7 was at the memory address given by the parameter, then the function should update that spot in memory to be 6.7. Afterward, **give an example** of calling *absval*.

void absval(double\* d) {

if (\*d < 0) {

\*d = \*d \* -1;

}

}

Example:

double val = -3.2;

absval(&val);

1. (11 pts) In this question, you will perform operations on a dynamic two-dimensional array. Consider the following variable declarations:

int rows, cols;

double\*\* table;

1. (1 pt) Ask the user to input the number of rows, and store the value in *rows*.

printf(“Enter rows: “);

scanf(“%d”, &rows);

1. (1 pt) Ask the user to input the number of columns, and store the value in *cols*.

printf(“Enter columns: “);

scanf(“%d”, &cols);

(Problem 5 is continued in the next page)

(Problem 5, continued)

1. (3 pts) Dynamically allocate memory so that *table* has space for a *rows* by *cols* matrix.

table = malloc(rows\*sizeof(int\*));

for (int i = 0; i < rows; i++) {

table[i] = malloc(cols\*sizeof(int));

}

1. (4 pts) Suppose all the elements in *table* have been initialized to some unknown values. Use your *absval* function from #4 to update each array element to be its own absolute value. (For example, if *table[0][2]* held -3.2, you should update it to hold 3.2.)

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

absval(&table[i][j]);

}

}

1. (2 pts) Free all the memory allocated in this problem.

for (int i = 0; i < rows; i++) {

free(table[i]);

}

free(table);