CIS330 **Name:** _____
Spring 2022
Midterm Exam
4/20/22, 10AM to 11:20AM
Time Limit: 80 Minutes

This exam contains 14 pages (including this cover page and extra sheets) and 6 questions. Total number of points is 100, excluding extra credit.

You are not allowed use any books, notes, calculators, or electronic devices of any kind. Write your answers **carefully** and **legibly**. Remember, **a partial answer is better than no answer**.

Feel free to skip around and go back to an earlier question later. You may find it helpful to skim over the entire exam **first** and start with the easier ones, then move on to the more difficult ones. Remember to distribute your time appropriately among the questions.

If you are only submitting the answers, please do not forget to write your name on the top of the first page.

There is an extra credit question at the end - do not miss it!
Good luck!

Grade Table (for instructor use only)

Question	Points	Score
1	8	
2	12	
3	20	
4	15	
5	30	
6	15	
Total:	100	

---

1 The C Language

1. (8 points) (7 minutes) Select the correction answers for the following multiple-choice questions.

   (a) (2 points) (1 minute) What would be the output of the following piece of C code?

```
int i = 5;
int j = 3;
int k = (j - i) ? ++i: j++;
printf("k is now %d\n", k);
```

   A. `k is now 5`
   B. `k is now 3`
   C. `k is now 6`
   D. `k is now 4`
   E. `k is now -2`
   F. None of the above.

   (b) (4 points) (4 minutes) Given the following piece of code, what will be printed?

```
int i[] = {5, 4, 3};
int* ip = &i[2];
int* jp = &i[1];
int* kp = &i[0];
printf("%d %d %d\n", *(ip--), *(++jp), *(kp+=2));
printf("%d %d %d\n", ++(*ip), *(jp--), ++(*kp));
printf("%d %d %d\n", *(--ip), ++(*jp), (*kp)++);
```

   A. 3 3 3
      5 3 4
      5 6 4

   B. 3 3 3
      4 4 4
      6 4 4

   C. 3 3 5
      4 4 4
      5 6 4

   D. 3 3 5
      5 3 4
      6 4 4

(c) (2 points) (2 minutes) Given the following code, which of the following is **not** a correct method for printing `A[i]`.

```
int* A = (int*) malloc(sizeof(int) * array_length);
```

A. `int* B = A;` `printf("%d\n", B[i]);`
B. `int* B = &(A[i - 2]);` `printf("%d\n", B[2]);`
C. `int* B = A + i ;` `printf("%d\n", *B);`
D. `int** B = &A;` `printf("%d\n", B[0][i]);`
E. All of the above are correct.

2. (12 points) (12 minutes) Answer the following short-answer questions.

(a) (6 points) (6 minutes) **At least** how many bytes are allocated in the heap memory for the following piece of code and its execution? You may get partial credit if you include the calculation.

```
#define MAX_NAME_LENGTH 100
#define MAX_NUM_COURSES 50

struct my_struct {
  int id;
  char* name;
  int* grade;
};

int main(int argc, char** argv)
{
  int num_students = atoi(argv[1]);
  struct my_struct* my_arr;
  my_arr = (struct my_struct*) malloc(sizeof(struct my_struct) * num_students);
  for(int i = 0; i < num_students; i++) {
    my_arr[i].name = (char*) malloc(sizeof(char) * MAX_NAME_LENGTH);
    my_arr[i].grade = (int*) malloc(sizeof(int) * MAX_NUM_COURSES);
  }

  return 0;
}
./a.out 10
```

(b) (2 points) (2 minutes) Given the following piece of code, what will be printed?

```
#define calcCircleArea(r) 3 * r * r
#define calcVolume(r,h) calcCircleArea(r) * (h)

int main()
{
  int r1 = 3;
  int r2 = 4;
  int h = 2;
  float volume = calcVolume((r1 + r2), h);
  printf("Volume is %f\n", volume);
  return 0;
}
```

(c) (4 points) (4 minutes) Describe the four components of `gcc`. Include a brief description of what each component does.

2 Coding in C Part 1

3. (20 points) (15 minutes) Given a 2-D array of integers (e.g., `int** arr`) with m rows and m columns, implement a function that **rotates** the array by 90 degrees **clockwise**.

   For example, for the given 3×3 matrix

1	2	3
4	5	6
7	8	9

   Its 90 degrees clockwise rotation would be

7	4	1
8	5	2
9	6	3

   Things to note:

   - You **must** use the function definition: `void rotate_clock(int** mat, int m);`
   - The rotated array must be stored in `int** mat`.
   - You **may** use additional storage (i.e., a temporary array) to do the rotation
   - However, make sure your function does not create memory leaks. **For this question, you must include code to free memory.**

   Hint: Think in terms of how rows are "moved."

Intentionally left blank

4. (15 points) (10 minutes) Given the following pieces of code, implement `init_2d()` and `free_2d()` functions that **allocates** and **frees** a 2-D array of "`int`". (10 minutes)

```
int main()
{
  int first_dim = 10;
  int second_dim = 20;
  int** my_array = NULL;

  init_2d(&my_array, first_dim, second_dim);
  // Do some stuff here with the 2-D array
  // ...
  free_2d(my_array, first_dim, second_dim);

  return 0;
}
```

   (a) (10 points) (7 minutes) Implement the `init_2d` function. Include the function definition as well. Even if some parameters are redundant, still include them to be consistent with how it is used in the `main` function.

(b) (5 points) (3 minutes) Implement the `free_2d` function. Make sure to free memory allocated to "`name`". Include the function definition as well. Even if some parameters are redundant, still include them to be consistent with how it was used in the `main` function.

3 Coding in C Part 2

5. (30 points) (20 minutes) An image is a 2-D array of pixels (or picture elements) and for a black-and-white image, a pixel value ranges from 0 to 255. Given the following code that implements an image:

```
typedef unsigned char pixel;
int main(int argc, char** argv)
{
  int rows = 30;
  int cols = 40;
  pixel** image = NULL;

  init_image(&image, rows, cols); // allocates memory & initializes to 0s
  load_image(image, rows, cols);  // load an image

  print_image(image, rows, cols); // print the image
  blur_image(image, rows, cols);  // blur the image
  print_image(image, rows, cols); // print the image to see if blurred correctly

  return 0;
}
```

   (a) (8 points) (5 minutes) Implement the `init_image()` function that allocates memory for a 2-D array of `pixel` and initializes all elements in the 2-D array to 0. Include the function definition and make sure the input parameters match what is shown in the `main()` function.

(b) (22 points) (15 minutes) Blurring an image can be done by going through each pixel in the image and replacing the pixel value with an *average* of pixel values of itself and its immediate north, east, west, and south neighbors. For example, given the following 4×4 image:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Blurring the image should produce the following new image:

2	3	4	5
5	6	7	7
9	10	11	11
12	13	14	14

For the pixel in position (2,1) (highlighted in light gray), the blurred pixel in the new image is calculated by:
(10 (itself) + 6 (north) + 11 (east) + 9 (west) + 14 (south)) / 5 = **10**
For the pixel in position (0,0) (highlighted in dark gray), the blurred pixel in the new image is calculated by:
(1 (itself) + 2 (east) + 5 (south)) / 3 = 2

Implement the `blur_image()` function that does this. Include the function definition and make sure the input parameters match what is shown in the `main()` function. **Do not** worry about memory leaks in this function.

(**HINT**: You may need to allocate a temporary 2-D array of pixels to do this correctly. You may call the init_image() function from 5.a above to do this.)

Intentionally left blank

6. (15 points) (10 minutes) Given an array of integers, we want to calculate its prefix-sum. However, instead of calculating one large prefix-sum from the entire array, we wish to do multiple prefix-sums over different segments of the array.

   That is, given an array of integers `A` and another array `F` consisting of 1s and 0s to indicate the beginning of each segment (1 indicates the beginning of a segment), we want to calculate the prefix-sums array `C` as shown below.

```
1  2  3  4  5  6     A
1  0  0  1  0  1     F
1  3  6  4  9  6     C
-------  ----  -
1        2     3     segment
```

   Notice that a prefix-sum is calculate for each segment.

   For example, for segment 1,

   - C[0] = A[0] = 1
   - C[1] = A[0] + A[1] = 3
   - C[2] = A[0] + A[1] + A[2] = 6

   For segment 2,

   - C[3] = A[3] = 4
   - C[4] = A[3] + A[4] = 9

   Things to note:

   - The algorithm must be **in-place** (i.e., only one integer swap variable is allowed).
   - Assume that the first element of array `F` will always be 1 to indicate the beginning of the first segment (i.e., `F[0] = 1`)
   - Use the function definition `void segmented_scan(int* A, int* F, int m)`, where `m` is the number of elements in the arrays `A` and `F`. The result must be stored in `A`.

Intentionally left blank

Extra Credit (5 points) Describe two advantages of dynamic library over static library (i.e., archive).