Name / ID (please PRINT) **Sequence #:_____** **Seat Number:____**

_____ _____

CS 2123.001 Data Structures

Spring 2018 – Midterm1 -- Feb 8, 2018
You have 75 min. Good luck.

- This is a **closed book/note** examination. But *You can use the reference card(s) given to you.*
- This exam has 5 questions in 9 pages. Please read each question carefully and answer all the questions, which have 100 points in total. Feel free to ask questions if you have any doubts.
- **Partial credit will be given, so do not leave questions blank**.

You can get **1pt bonus** credit if you complete the **boldfaced column** of the following table. Please do this after answering all the questions in the exam. You will also get **1pt bonus** if the total expected score is within ±5 of total received score.

Question	Topic	Possible Points	**Student Expects to receive out of**	Student's Received Score
1	String processing/manipulation	20	**/20**	
2	Pointers (tracing a program)	20	**/20**	
3	2D Arrays (static size)	20	**/20**	
4	Command Line Arguments	20	**/20**	
5	Files, Dynamic Memory and Struct	20	**/20**	
	Bonus If this table is completed	1		
	If the total expected score is within ±5 of total received score	1		
	If you completed the survey on BB Learn	3		
Total		100+5		

1. (20 pt) Implement a function `char *lastname_firstname(char *FN, char *LN);` which dynamically creates a new string (`ns`) by merging the given `"FirstName"` (FN) and `"LastName"` (LN) in the format of `"LastName, FirstName"`

   For example:
   ```
   char *ns;
   ns = lastname_firstname("Turgay", "Korkmaz");
   ```
   `ns` should be pointing to a dynamically created new string **`"Korkmaz, Turgay"`**

   As shown below, you can use **only** `strlen(char *s)` function from the standard string.h library. You are **not** allowed to use any other standard string functions like `strcpy()`, `strcat()`, `sprintf()` etc.... So, your code should do all the copying, formatting etc.

```
char * lastname_firstname(char *FN, char *LN); {
                                    /* you can use either pointer or array notation */
  char *ns;
  int lenFN, lenLN, i, j;

  lenFN = strlen(FN);
  lenLN = strlen(LN);
```

2. (20 pt) **Trace** the following program, **show how values change** in memory, and **give the output.**

```
#include <stdio.h>
typedef struct {
   int x;
   int y;
} fractionT;
main()
{
  int z[6] = {3, 4, 5, 7, 2, 9};
  int *p1, **p2;
  fractionT f[2];

  p1 = &z[3];

  p2 = &p1;

  *p1-- = 9;

  *--p1 = 8;

  printf("%d %d %d %d %d \n",
         z[0], z[1], z[2], z[3], **p2);

  z[5] = func(&f[1], &f[1].y, &p1);

  printf("%d %d %d %d \n",
         z[3], z[5], f[0].y, f[1].y);

}

int func(fractionT *a, int *b, int **c)
{
   int x=5, y=24;

   *b = y / x % 3;

   a--;

   *c = &a->y;

   **c = 13;

   return *b + a->y;
}
```

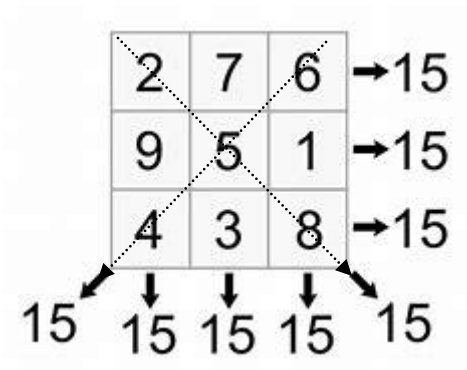
name	Address	MEMORY Content/Value
z[0]	12	
z[1]	16	
z[2]	20	
z[3]	24	
z[4]	28	
z[5]	32	
p1	36	
p2	40	
f[0].x	44	
f[0].y	48	
f[1].x	52	
f[1].y	56	
	100	
a	104	
b	108	
c	112	
x	116	
y	120	
	124	

OUTPUT

3. (20 pt) Recall the Sudoku puzzle that you studied in Assignment 1. In this question, you will work on a similar number placement puzzle known as Magic Square (MS).

   [Here is the description from the web] A magic square is an arrangement of distinct integer numbers (i.e., each number is used once) from 1 to N^2 in an NxN square grid, where the numbers in each **row**, and in each **column**, and the numbers in the **main** and **secondary** diagonals, all add up to the same number, which is $N*(N^2 + 1) /2$.

   For example, we can place the numbers from 1 to $9 = 3^2$ on a 3x3 MS with the same sum of 15, and place the numbers from 1 to $25=5^2$ on a 5x5 MS with the same sum of 65, as shown in the below figures.



23	6	19	2	15
4	12	25	8	16
10	18	1	14	22
11	24	7	20	3
17	5	13	21	9

   Suppose we are interested in checking if a given `int MS[N][N] = {{/*initial values */}, ...};` is a valid Magic Square or not. Also suppose we have the four helper functions in the next page that respectively check if the sum of the numbers in each **row**, and in each **column**, and in the **main** and **secondary** diagonals are all equal to the same sum (SS) = $N*(N^2 + 1) /2$. Then we can simply check if a given `MS[N][N]` is a valid Magic Square or not as follows:

```
/* suppose all standard C libraries are included here */
#define N   3      /* the number N can be large in an actual program */

void main()
{
  int MS[N][N] = {{2,7,6},
                  {9,5,1},
                  {4,3,8}};
  int SS = (N*(N*N+1)/2);

  if(allRowsOK(MS, SS)  && allColumnsOK(MS, SS) &&
     mainDiagOK(MS, SS) && secondaryDiagOK(MS, SS) )
     printf("YES, this is a Magic Square! \n");
  else
     printf("NO, this is NOT a Magic Square! \n");
}
```

/* NOW You are asked to **implement the four helper functions** in the next page! Actually, one is already implemented for you as a sample! */

```
/* This function returns 1 if the sum of
every row is equal to SS;
otherwise, return 0 */                // 8pt

int allRowsOK(int MS[][N], int SS)
{
  int sum, i, j;
```

```
/* This function returns 1 if the sum of
every column is equal to SS; otherwise,
return 0 */                           // 7pt

int allColumnsOK(int MS[][N], int SS)
{
  int sum, i, j;
```

```
/* This function returns 1 if the sum of
the main diagonal is equal to SS;
otherwise, return 0 */

int mainDiagOK(int MS[][N], int SS)
{
  int sum, i;

  sum=0;
  for(i=0; i < N; i++){
     sum = sum + MS[i][i];
  }
  if (sum != SS) return 0;
  return 1;
}
```

```
/* This function returns 1 if the sum of
the secondary diagonal is equal to SS;
otherwise, return 0 */                // 5pt

int secondaryDiagOK(int MS[][N], int SS)
{
  int sum, i;
```

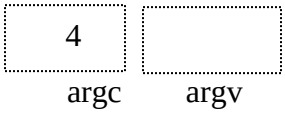
4. (20pt) Write a program that finds the **sum** of the integer numbers given as **command line arguments**. User may give as many numbers as she wants. If no number is given, print 0. Also assume that user will always type numbers, so don't worry about arbitrary input.
Hint: Recall that `int atoi(char *str);` converts String to Integer.

Here is an examples:

```
> sum_prog 300  400  200
900
> sum_prog
0
```

a. (10pt) Conceptually draw the memory representation or snapshot when the user calls your program as

```
> sum_prog 300  400  200
```



[Part (b) of this problem is in the next page.]

b. (10pt) Complete the following program to find and print the sum of the numbers given as command line arguments.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
```

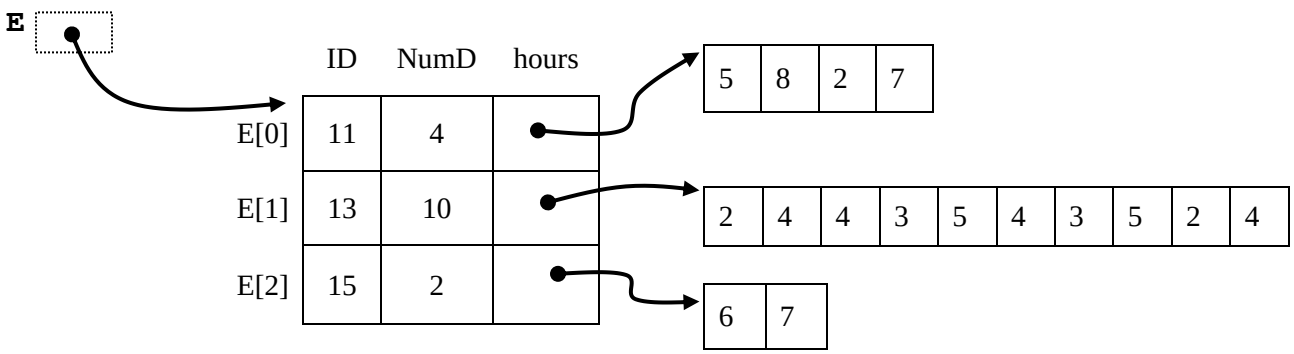
5. (20pt) This question uses almost the same file format as in Quiz 6 that we solved in class. But this time, instead of reading and summarizing the data from the input file, we would like to create a dynamic data structure to store all the information about the employees in the memory for further analysis.

   Suppose the employee data file (say emp.txt) is now starting with an integer showing *the number of employees* in the file. It then contains that many lines, which are formatted as in Quiz 6. So the file has the followings in each line per employee: *employee ID, how many days he/she worked*, and *how many hours he/she worked in each day*. All these values are integers.

   Here is a sample file with 3 employees:

```
3
11   4     5 8 2 7
13  10     2 4 4 3 5 4 3 5 2 4
15   2     6 7
```

   Complete the following program that can **read** emp.txt file and **create** the dynamic structure as shown in the below figure. There is no output file.




```
/* suppose all standard libraries are included here */

typedef struct {
  int ID;
  int NumD;
  int *hours;  /* a dynamic array to store # of hours this employee worked in each day.*/
} empT;

#include <stdio.h>
int main(void)
{
  FILE *infp;
  int NumEmp, ID, NumD, hour;
  int i, j;          /* if needed you can declare other variables here */
  empT *E;
```

```
if ((infp = fopen("emp.txt", "r"))==NULL){
    printf("Input file cannot be opened\n");
    return -1;
}
if(fscanf(infp, "%d", &NumEmp)!=1) exit(0);
```