Project 2
150 – 200 lines of code

Sports analysis

You've decided to analyze the results of the last 4 years of your college [sports/debate/robotics] team. You have data files that list the results of every event of the 10 schools in your division from the last several years. Write a program that can answer the following questions:

1. How many events did each team win each year?
2. How many total points did each team score each year?
3. What is the top team each year?

Learning Objectives covered:

1. I can write and compile simple C programs that communicate with the terminal and contain functions, loops, and conditionals.
2. I can write a program that adheres to a style guide.
3. I can write high-quality documentation.
4. I can write a program that reads from files and writes formatted output to files.
5. I can write programs using 1D and 2D arrays with functions.
6. I can write a program that uses command-line arguments.
7. I can write and use a `makefile`.

Input files:
Each input file is formatted as follows where column 0 is the away team, column 1 is the home team, and columns 2 and 3 are the scores.

```
Year: 2022
  1   2  47  46
  1   3  22  39
```

You should use the first line of the file to get the year.
You can assume each file contains data for 10 teams (1 to 10) with a total of 45 lines of data.

Program structure:
Four code files:

- `analyze_scores.c`
  - `main()`
- `analysis_functions.c`
  - 3 or more functions, at least 1 function must take a 2D array as a parameter
- `analysis_function.h` with documentation and prototypes
- `makefile`

Before you start programming, thoroughly read the program description and outline your program using pseudocode. You should separate the program into multiple functions.

Compiling and running:
Your program will be compiled using your `makefile`.
Your program should be run with the following command line arguments where the last command line argument is the name of the output file and arguments 1 to N-1 are the names of the input files.

```
./runScores scores20.txt scores21.txt ... results.txt
```

Output:
Your program should produce the following output and files (the top team is based on meets won, not total score):

Terminal Output

```
The top team of 2019 was Team 10.
The top team of 2020 was Team 10.
The top team of 2021 was Team 5.
```

With one line for each file listed at the command line

Output file format:

```
Team 2019 2020 2021 2019 2020 2021
   1    4    4    3  216  317  202
   2    4    5    6  286  278  266
```

where columns 1-4 contain the # of winning meets for the given team, and columns 5-8 contain the total points earned for each year.

Additional details:

1. Check that the user has included enough command line arguments. If the user uses incorrect command line arguments, print the following:

```
Run the program with a list of input file, followed by the output file

./runScores file_name.txt ... results.txt

where file_name.txt ... is a list of files to process
and results.txt is the name of the output file to produce.
```

2. Managing data:
    a. Read the input files into a 2D array
    b. Process the data by calling `findWins` and `findTotal`
    c. After reading and processing each input file, print the output to the terminal
    d. Store the total scores for each team and year in a 10 x N array where N is the number of years
    e. Store the # of winning meets for each team and year in a 10 x N array where N is the number of years
    f. After reading and processing all years, print the results to the output file
3. If two teams tie in a game, mark them both as winning the game.
4. The top team is the team with the most wins.
5. If two teams are tied for top, then the top team is the one with the most points. If that is also a tie, declare it a tie:
    Ex: Team 2 has won 7 games with 57 points. Team 7 has won 7 games with 55 points. Team 2 is the top team
    Ex: Team 4 has won 9 games with 94 points. Team 9 has won 9 games with 94 points. The teams tied. Output the following:

```
The top team of 2023 was tied between Team 4 and Team 9.
```

Example functions:
You are welcome to write any functions that are useful in your program. If you are struggling to separate the tasks into functions, here are some ideas you are welcome to use.

```
int load_data(int numRows, int numCol, int teamData[][numCol], char*
fileName);
// loads the data from fileName into teamData. Returns the year.

void findTotals(int numTeams, int totals[], int dataRows, int dataCol, int
team_data[][dataCol]);
// Loops through teamData, and stores the total scores of each team in
totals[]

void findWins(int numTeams, int wins[], int dataRows, int dataCol, int
team_data[][dataCol]);
// Loops through team_data, and stores the number of wins of each team wins[]

int findTop(int numRows, int wins[], int totals[], int is_top[]);
// Uses the data in wins[] and totals[] to find the top winner for the year.
Marks that team as the winner in the is_top[] array (mark multiple if
multiple teams win). Returns a flag of 1 or 2 to indicate if there is a tie.
```

Restricted material:
You may not use any of the following programming concepts in this assignment:
- structs - global variables - static

Deliverables:

1. Write a document called **README.txt** that summarizes your program, how to run it, and detailing any problems that you had. If you used any outside resources for this assignment, be sure to cite your sources *and* explain in detail how the code works.
2. `analyze_sales.c`, `sales_functions.h` and `sales_functions.c`.
3. Your `makefile`. The executable must be called `runScores`

Using your resources:
If you consult web resources or other students or staff when developing your program, *you must cite your source*. If you completed the entire program on your own, you should say in your write up file that this is entirely your work.

Grading:
The full grading rubric is explained in the CS 2303 Project Guidelines document. Five additional rubric items are listed below.

Met	Rubric Item
	Program does not contain restricted items
	Program contains multiple source code files with correct #include statements
	Makefile contains 2 or more keywords
	Program correctly passes a 2D array to a function
	Function prototypes are in the .h file and it is included correctly in the .c files

The grading breakdown is shown in the following table. To earn a particular grade, you must meet all criteria in the row corresponding to that grade. There are 7 core rubric items and 10 supplemental items (5 in the Rubric document and 5 listed above).

		Rubric Items Met	
	Autograder tests	**Core**	**Supplemental**
Excellent	All Passed	7	9+
Meets Expectations	All Passed	7	8+
Needs Revision	All Passed	N/A	N/A
Not Graded	Failed 1 or More test	N/A	N/A

Passing 2D arrays:
Passing a 1D array to a function is quite simple; we only have to put an array variable in the function prototype and declaration, along with a second variable to store the size of the array, as shown here:

```
int function_1(int size, int data[]);
```

or here:

```
int function_1(int data[], int size);
```

To call this function, simply pass it the name of the array:

```
int x = function_1(5, myData);
```

Two-dimensional arrays require the compiler to know the number of columns in the array (this is part of how the compiler finds the array index). Because of this, the column variable needs to be in the array variable, as shown here:

```
int function_2(int rows, int columns, int data[][columns]);
```

To call this function, simply pass it the name of the array:

```
int y = function_2(5, 6, myData2D);
```

```
int function_1(int size, int data[]);
```

The following line of code will not work because `columns` is not declared before `data[][columns])`

```
int function_2(int data[][columns], int rows, int columns);
```