CSE 220: Systems Fundamentals I Spring 2018
Homework #2
Assignment Due: Saturday March 3, 2018 by 11:59 pm via Sparky

⚠ **READ THE WHOLE DOCUMENT TWICE BEFORE STARTING!**

⚠ DO **NOT** COPY/SHARE CODE! We will check your assignments against this semester and previous semesters!

ℹ Download the Stony Brook version of MARS posted on Piazza. **DO NOT USE** the MARS available on the official webpage. The Stony Brook version has a reduced instruction set, added tools, and additional system calls you will need to complete the homework assignments.

⚠ You personally must implement the assignment in MIPS Assembly language by yourself. You may not use a code generator or other tools that write any MIPS code for you. You must manually write all MIPS Assembly code you submit as part of the assignment. You may also not write a code generator in MIPS Assembly that generates MIPS Assembly.

⚠ All test cases MUST execute in 10,000 instructions or less. Efficiency is an important aspect of programming.

⚠ Any excess output from your program (debugging notes, etc) WILL impact your grading. Do not leave erroneous printouts in your code!

⚠ Do not submit a file with the functions/labels `main` or `_start` defined. You are also not permitted to start your label names with two underscores (`__`). You will obtain a ZERO for the assignment if you do this.

Introduction

In this assignment you will be creating functions and working with 1D and 2D arrays. The goal is to understand function calls, returning values, and the role of register conventions. We will implement functions to gather information about the students in a class (1D array) and then based on a 2D seating chart of students write a function to identify potential student cheaters.

You **MUST** implement all the functions in the assignment as defined. It is OK to implement additional helper functions of your own in `hw2.asm`.

ℹ If you are having difficulties implementing these functions, write out the pseudo code or implement the functions in a higher-level language first. Once you understand the algorithm and what steps to perform, then translate the logic to MIPS.

ℹ When writing your program, try to comment as much as possible. Try to stay consistent with your formatting. It is much easier for your TA and the professor to help you if we can figure out what your code does quickly.

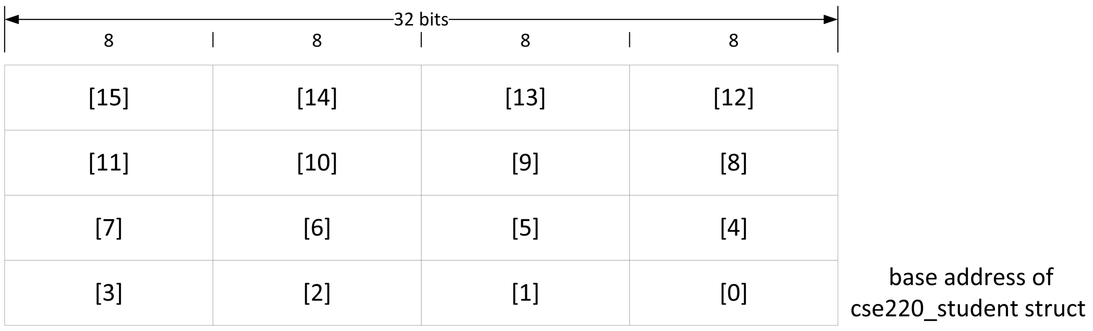
For this assignment, there is a new `cse220_exam` struct that will be used in addition to the old `cse220_student` struct.

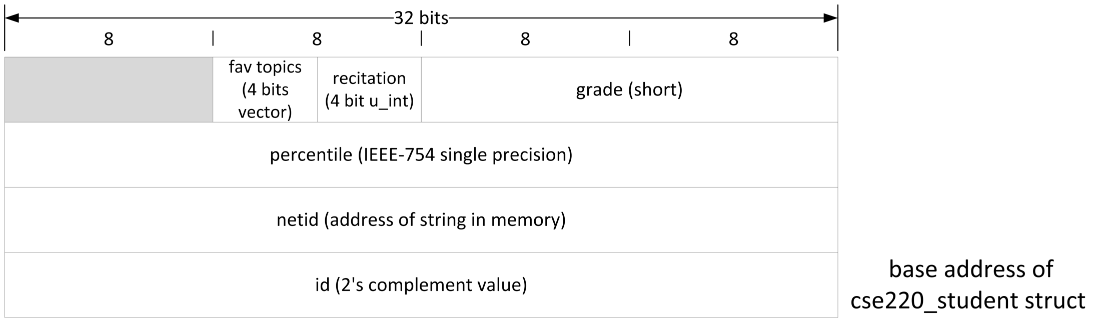
```
struct cse220_student {
  int      id;           // size: 4b (b == bytes)
  String   netid;        // size: 4b; starting address of the string
  float    percentile;   // size: 4b; single precision IEEE-754
  short    grade;        // size: 2b; byte[0] is letter and byte[1] is sign
  nibble   recitation;   // size: 4 bits; NOTE: nibble isn't a true C datatype
  nibble   favtopics;    // size: 4 bits; bit vector
};

struct cse220_exam {
  cse220_student* student;  // size: 4b; starting address of cse220_student struct
  int             grade;    // size: 4b; exam grade of the student
};
```

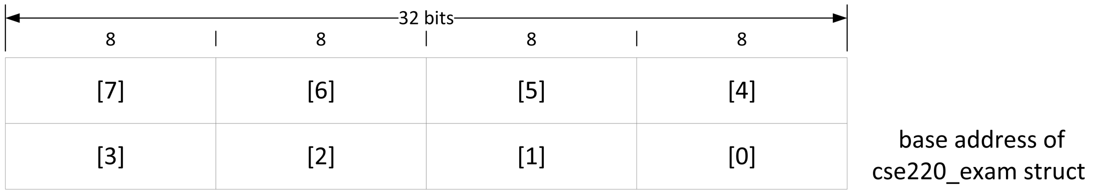
The `*` notation used in the `cse220_exam` struct is known as a pointer in C. A pointer is a space in memory which holds a memory address. The data type associated with the pointer identifies the how the data should be accessed at the memory location that is specified by the pointer.

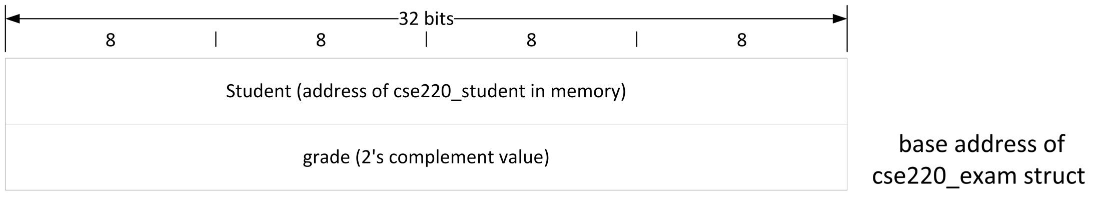
Visually, the structs are defined in memory as:





32 bits
8
8
8
8
fav topics (4 bits vector)
recitation (4 bit u_int)
grade (short)
percentile (IEEE-754 single precision)
netid (address of string in memory)
id (2's complement value)
base address of cse220_student struct





32 bits
8
8
8
8
Student (address of cse220_student in memory)
grade (2's complement value)
base address of cse220_exam struct

Getting started

Download `hw2.zip` from Piazza in the Homework section of Resources. This file contains `hw2.asm` and multiple `hw2_main` files, which you can use to test for the assignment. At the top of your `hw2.asm` program in comments put your name and SBU ID number.

```
# Homework #2
# name: MY_NAME
# sbuid: MY_SBU_ID
```

The `hw2_main` files include a `gradeshelpers.asm` file which has two provided functions `getGradeIndex` and `getGrade`. These functions map between a grade and an assigned index value. You **MUST** treat these functions as black boxes, meaning do not rely on their implementations in any way other than their specification.

⚠ We will modify the implementations of these functions during grading!

⚠ Do NOT include the `gradeshelpers.asm` file in your `hw2.asm` file! Your program will not compile with our test cases and you will get a ZERO.

- `int getGradeIndex(short grade)`

  The function takes the 2 characters for the grade as an argument and returns the assigned index associated with the grade. It is best to treat this function as a blackbox, as its implementation should not matter to your function.

  Function parameter and return value summary:

  - `grade`: the grade in the set {"A ", "A-", "B+", "B ", "B-", "C+", "C ", "C-", "D+", "D ", "D-", "F " }
  - *returns*: index value associated with specified grade, -1 if invalid grade

- `short getGrade(int index)`

  The function takes a given index and returns the associated grade. It is best to treat this function as a blackbox, as its implementation should not matter to your function.

  Function parameter and return value summary:

  - `index`: value to look up associated grade
  - *returns*: 2 characters for the grade associated with the index, -1 if `index` is outside of range [0,11]

How to test your functions

To test your functions, simply open the provided `hw2_main` file which tests your function in MARS. Next, assemble the `main` file and run. Mars will take the contents of the file referenced with the `.include` at the end of the file and add the contents of your `hw2.asm` file before assembling it.

The main files only call each function with ONE of the sample test cases and print the return value(s) to the output. You will need to modify the arguments passed to the functions to test with the other cases.

⚠ It is highly advised to write your own main programs (new individual files) and create your own data arrays (in the main files) to test each of your functions thoroughly. Your assignment will not be graded using the examples provided!

⚠ Make sure to initialize all of your values within your functions! Never assume registers or memory will hold any particular values!

Any modifications to the `main` files will not be graded. You will only submit your `hw2.asm` file via Sparky. Make sure that all code required for implementing your functions are included in the `hw2.asm` file!

⚠ There is no need to define a `.data` section within your homework. Your functions should not refer to any labels directly.

To make sure that your code is self-contained, try assembling your `hw2.asm` file by itself in MARS. If you get any errors (such as a missing label), this means that you need to refactor (reorganize) your code, possibly by moving labels you inadvertently defined in a `main` file to `hw2.asm`.

Part 1: Basic Leaf Functions

In this section you will create "leaf" functions (a function which does not call any other functions).

a. `int recitationCount(cse220_student[] class, int classSize, int rnum)`

The function counts the number of students in `class` which are in recitation `rnum`.

ℹ We WILL NOT test the function with an invalid `class` argument address.

Function parameter and return value summary:

- `class` : starting address of the 1D array of students in the class.
- `classSize` : number of entries in class array.
- `rnum` : recitation number.
- *returns*: the number of students in the class who are in the specified recitation.

Return -1 for error in any of the following cases:

- `classSize` ≤ 0.
- `rnum` is not in the set {8, 9, 10, 12, 13, 14}.

❗ Do not modify `class` in memory.

Examples:

Code	Return Value
`recitationCount(class1, 0, 8)`	-1
`recitationCount(class1, 5, 11)`	-1
`recitationCount(class1, 5, 8)`	2
`recitationCount(class2, 5, 12)`	3
`recitationCount(class2, 10, 12)`	5

ℹ The length specified for an array does not have to match the actual length of the array. By setting the value smaller, we can check subsets of the array entries.

b. `float aveGradePercentage(int[] histogram, float[] gradepoints)`

The function calculates the average grade percentage based on the specified histogram counts in `hist`. The `hist` array contains the number of students who obtained a given grade.

ℹ We WILL NOT test the function with an invalid `histogram` or `gradepoints` argument address, and each will always refer to 12 entries.

Function parameter and return value summary:

- `histogram` : address of 1D array of integers specifying the number of students with each grade.
- `gradepoints` : address of 1D array of floats specifying the gradepoint values.
- *returns*: average grade percentage, -1 if error.

Return (float -1) for error in any of the following cases:

- any entry of `histogram` or `gradepoints` contains a negative value.
- `histogram` contains a total count of 0 (ie. there are no grades specified).

ℹ The bit representation for single precision floating point -1 is 0xBF800000.

ℹ Reminder, even though you are returning a float, place the return value in $v0.

Examples:

Code	Return Value
`aveGradePercentage(histErr1, gradepoints1)`	-1.0
`aveGradePercentage(histErr2, gradepoints1)`	-1.0
`aveGradePercentage(hist1, gradepointsErr)`	-1.0
`aveGradePercentage(hist1, gradepoints1)`	2.4
`aveGradePercentage(hist1, gradepoints2)`	2.6029413
`aveGradePercentage(hist2, gradepoints1)`	3.5066667
`aveGradePercentage(hist2, gradepoints2)`	4.4.266667

c. `float favtopicPercentage(cse220_student[] class, int classSize, nibble topics)`

The function calculates the percentage of students in the class who like ANY of the topics specified by `topics` (1 means consider, 0 means do not consider).

Function parameter and return value summary:

- `class` : starting address of the 1D array of students in the class.
- `classSize` : number of entries in class array.
- `topics` : bit vector specifying topics to consider.
- *returns*: percentage of students who like topics, -1 if error.

ℹ You will be returning the float in $v0 even though it is not a floating point register.

Return (float -1) for error in any of the following cases:

- `classSize` ≤ 0.
- `topics` is not in range [1,15].

ℹ The bit representation for single precision floating point -1 is 0xBF800000.

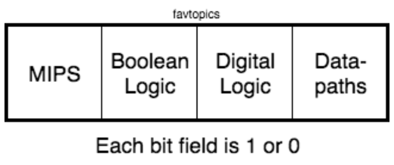
Examples:

Code	Return Value
`favtopicPercentage(class1, 0, 2)`	-1.0
`favtopicPercentage(class1, 5, 0)`	-1.0
`favtopicPercentage(class1, 5, 1)`	0.2
`favtopicPercentage(class1, 5, 12)`	0.6
`favtopicPercentage(class2, 5, 15)`	1
`favtopicPercentage(class2, 10, 2)`	0.6

d. `int findFavtopic(cse220_student[] class, int classSize, nibble topics)`

The function determines the favtopic of the class amongst the ones specified in `topics`. `topics` is a bit vector specifying which of the topics to consider when calculating the favorite topic (1 means consider, 0 means do not consider). In the case of a tie, return the higher valued bit topic (eg. MIPS > Boolean Logic > Digital Logic > Datapaths).

Function parameter and return value summary:



- `class`: starting address of the 1D array of students in the class.
- `classSize`: number of entries in class array.
- `topics`: bit vector specifying topics to consider for favorite topic.
- *returns*: a 1 in the bit position corresponding to the favtopic, -1 if error.

Return -1 for error in any of the following cases:

- `classSize` $\leq$ 0.
- `topics` is not in range [1,15].
- All students have no favtopics from the `topics` (eg. there is no favorite as the count is 0 for all topics)

Examples:

Code	Return Value
`findFavtopic(class1, -2, 6)`	-1
`findFavtopic(class1, 5, 0)`	-1
`findFavtopic(class1, 5, 5)`	4
`findFavtopic(class1, 5, 15`	2
`findFavtopic(class1, 5, 12)`	8
`findFavtopic(class2, 8, 11)`	8
`findFavtopic(class2, 10, 15)`	1

Part 2: Non-Leaf functions

In this part of the assignment you will create more complex functions which make function calls. These functions should follow MIPS register conventions using prologue and epiloge blocks.

e. `(int,int) twoFavtopics(cse220_student[] class, int classSize)`

The function returns the top 2 favorite topics amongst the students in the course. In the case of a tie, the function return the higher valued bit topic (eg. MIPS > Boolean Logic > Digital Logic > Datapaths).

❗ Your function MUST CALL `findFavtopic`.

- `class`: starting address of the 1D array of students in the class.
- `classSize`: number of entries in class array.

- *returns*: (favorite, second favorite) or (-1,-1) on error. Each favorite is specified by a 1 in the bit position corresponding to the favtopic

Return (-1,-1) for error in any of the following cases:

- `classSize` ≤ 0.
- All students have no favtopics from the `topics` (eg. there is no favorite as the count is 0 for all topics)

Examples:

Code	Return Value
`twoFavtopics(class1, -2)`	(-1, -1)
`twoFavtopics(class1, 5)`	(2,8)
`twoFavtopics(class2, 7)`	(8, 2)
`twoFavtopics(class2, 10)`	(1, 2)

f. `float calcAveClassGrade(cse220_student[] class, int classSize, int[] histogram, float[] gradepoints)`

The function creates a `histogram` count of the cse220_student `grade` fields in the `class` and calculates the average grade percentage for the course.

You **MUST** use the `int getGradeIndex(short grade)` function provided to determine which index in `histogram` to place each grade count. Treat this function as a blackbox!

We WILL NOT test the function with an invalid `histogram` or `gradepoints` argument address, and each will always refer to 12 entries.

Remember, the grade stored in the struct has a space character (ASCII 32), if it does not have a '+' or '-' modifier.

Function parameter and return value summary:

- `class` : starting address of the 1D array of students in the class.
- `classSize` : number of entries in class array.
- `histogram` : starting address of a 1D array in memory to hold the histogram counts.
- `gradepoints` : 1D array of float specifying the gradepoint values.
- *returns*: average grade percentage or -1 if error.

Return (float -1) for error in any of the following cases. When an error occurs, the values stored to histogram are considered invalid.

- `classSize` ≤ 0.
- `class` contains an invalid grade (eg. `getGradeIndex` returns -1).

The bit representation for single precision floating point -1 is 0xBF800000.

Reminder, even though you are returning a float, place the return value in $v0.

The function must modify `histogram` in memory. `histogram` may already contain values. Do not assume the memory is initialized to 0.

Your function MUST CALL `aveGradePercentage` and `getGradeIndex`.

Examples:

Code	Return Value
`calcAveClassGrade(class1, -12, hist, gradepoints1)`	-1.0
`calcAveClassGrade(class1, 5, hist1, gradepoints1)` `hist` after function call contains [1,0,0,1,1,0,0,1,0,0,0,1]	2.28
`calcAveClassGrade(class2, 10, hist, gradepoints1)` `hist` after function call contains [0,2,1,1,0,3,0,0,1,0,1,1]	2.2599998

g. `int updateGrades(cse220_student[] class, int classSize, float[] cutoffs)`

The function compares each student's `percentile` to the `cutoffs` values and reassigns the student a new `grade`. The `cutoffs` array has the same index mapping as specified in the `aveGradePercentage` function. Use the `getGrade` function to get the grades using the index used for `cutoffs`. Each cutoff value is inclusive, ie. if `cutoffs[0]` is 92.5 then any student with a percentile $\geq$ 92.5 is assigned an A.

You **MUST** use the `short getGrade(int index)` function provided. Treat this function as a black-box.

We WILL NOT test the function with an invalid `cutoffs` argument address, and `cutoffs` will contain 12 entries.

Function parameter and return value summary:

- `class`: starting address of the 1D array of students in the class.
- `classSize`: number of entries in class array.
- `cutoffs`: starting address of the 1D array of float values for each grade.
- *returns*: 0 if success, -1 if error

Return -1 for error in any of the following cases:

- `classSize` $\leq$ 0.
- `cutoffs` contains non-decreasing values, ie. index 0 has value 85 and index 1 has value 90.
- `cutoffs[11]` does not contain 0.

The function must modify `class` in memory.

Examples:

Code	Return Value
`updateGrades(class1, -2, cutoffs1)`	-1
`updateGrades(class1,  5, cutoffsErr)`	-1
`updateGrades(class1,  5, cutoffs1)` `class1` grades stored in structs after function call are A, B+, F, B+, C+	0
`updateGrades(class1,  5, cutoffs2)` `class1` grades stored in structs after function call are A, A-, F, A-, C+	0
`updateGrades(class2, 10, cutoffs1)` `class2` grades stored in structs after function call are D+, B, B-, B+, B-, B-, C, B+, C, A	0
`updateGrades(class2, 10, cutoffs2)` `class2` grades stored in structs after function call are D+, B+, B, A-, B, B-, C-, A-, C-, A	0

Part 3: 2D Arrays

h. `(int, int) find_cheaters(cse220_exam[][] seats, int rows, int cols, String[] cheaters)`

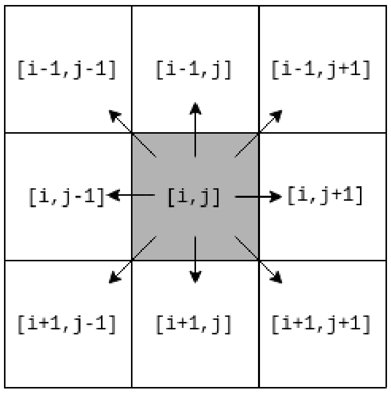
This function will use a "seating chart" to figure out which students cheated in the exam. The seating chart will be a `rows` x `cols` 2D array of `cse220_exam` structs.

ℹ Take a look at the introduction to see the details about the `cse220_exam` struct.

ℹ We guarantee that the `seats` array will only contain valid data. You do not need to check for valid values.

Iterate through the 2D array in row-major order from `(0, 0)` to `(rows-1, cols-1)`, checking if a student has the same exam score as any student in the adjacent (including diagonals) seats.

⚠ Watch out for boundaries of the room!



If you find a potential cheater, add the ADDRESS of the `netid` of the student you are currently checking to the list of strings specified by `cheaters`. You can access a student's information in the `cse220_student` struct using the address available in the `cse220_exam` struct.

ℹ It is guaranteed there is enough space allocated in memory to store the addresses of all strings in the `cheaters` argument.

All seats do not have to be filled. An empty seat will be denoted by a zero-ed out `cse220_exam` struct. All `students` may not be present for the exam and therefore may not appear in `seats`.

Function parameter and return value summary:

- `seats`: starting address of a 2D array of `cse220_exam` structs.
- `rows`: the number of rows in `seats`.
- `cols`: the number of cols in `seats`.
- `cheaters`: starting address of the array of strings where all the cheaters' `netid` will be stored.
- *returns*: (The number of cheaters, the number of students who took the exam) or (-1,-1) if error

Returns (-1, -1) for error in any of the following cases:

- `rows` ≤ 0
- `cols` ≤ 0

Examples:

Use `hw2_main_cheaters` to test this function. `cheaters.asm` contains the `.data` section. The

`Room1.png` and `Room2.png` images provide a visual of each test case and the resultant `cheaters` array (in terms of strings, not addresses).

Code	**Return Value**
`find_cheaters(room1, -3, 4, cheaters)`	(-1, -1)
`find_cheaters(room1, 3, -4, cheaters)`	(-1, -1)
`find_cheaters(room1, 3, 4, cheaters)`	(6, 12)
`find_cheaters(room1, 4, 3, cheaters)`	(6, 12)
`find_cheaters(room2, 3, 10, cheaters)`	(7, 14)
`find_cheaters(room2, 5, 6, cheaters)`	(4, 14)

Hand-in instructions

Do not add any miscellaneous printouts, as this will probably make the grading script give you a zero. Please print out the text exactly as it is displayed in the examples, one output line ONLY.

See Sparky Submission Instructions on Piazza for hand-in instructions. There is no tolerance for homework submission via email. They must be submitted through Sparky. Please do not wait until the last minute to submit your homework. If you are struggling, stop by office hours.

When writing your program try to comment as much as possible. Try to stay consistent with your formatting. It is much easier for your TA and the professor to help you if we can figure out what your code does quickly.