Jesus Ugarte

Exercise dynamic memory allocation

1.) Consider you want to dynamically allocate memory for an array of arrays, where each array length is variable. The type of the arrays would be float. Write a function that takes two items: i) an integer P that represents number of arrays, and ii) An array called Lengths of size P that contains Lengths for P number of arrays. After allocating memory for the arrays, the function should fill-up the arrays with random numbers from 1 to 100 (you can put an int to a float variable. The integer will get promoted to float automatically). At the end the function returns appropriate pointer. Next write a set of statements to show how you would free-up the memory.

In the blank bellow, write the appropriate return type.

a) float ** AllocateArrayOfArrays(int P, int *Lengths) {

```
float **arrays = (float **) malloc (sizeof(* lenght));
int i,j;
for (i=0; i< P ; i++)
{ arrays[i]= (float *) malloc (sizeof (float));

    for (j=0; j<*lenght ; j++)
    {
      arrays[i][j] = (int) malloc (size of (float));
    }arrays[i][j] = rand()%100+1;
}return arrays;
}
```

b) Write few C statements to declare an appropriate variable and call the AllocateArrayOfArrays() with example parameters. Later on write necessary C statement(s) to free the allocated memory.

```
Int main (void)
{
  Int P=2, Lenghts=3;

  float ** arrays = AllocateArrayofArrays( P, &lengths);
  releaseMemory(arrays, P, &lenghts);
  return 0;
}
```

```
Void releaseMemory(float ** arrays, P, *lengths)
{
    for(int i=0; i<P; i++)
    {
        for(Int J=0; J< *lenghts; J++)
        {
            free(arrays[i][J]);
        }
        free(arrays[i]);
    }
    free(arrays);
}
```

2. This problem relies on the following struct definition:

```
typedef struct Employee
{
        char *first; // Employee's first name.
        char *last; // Employee's last name.
        int ID; // Employee ID.
} Employee;
```

Consider the following function, which takes three arrays – each of length n – containing the first names, last names, and ID numbers of n employees for some company. The function dynamically allocates an array of n Employee structs, copies the information from the array arguments into the corresponding array of structs, and returns the dynamically allocated array.

```
Employee *makeArray(char **firstNames, char **lastNames, int *IDs, int n)
{
        int i;
        Employee *array = (Employee *)malloc(n * sizeof(struct Employee));
        for (i = 0; i < n; i++)
        {
                array[i].first = malloc(sizeof(char));
                array[i].last = malloc(sizeof(char));
                strcpy(array[i].first, firstNames[i]);
                strcpy(array[i].last, lastNames[i]);
                array[i].ID = IDs[i];
        }
        return array;
}
```

a) Fill in the blanks above with the appropriate arguments for each malloc() statement.

b) Next, write a function that takes a pointer to the array created by the makeArray() function, along with the number of employee records in that array (n) and frees all the dynamically allocated memory associated with that array. The function signature is as follows:

```
void freeEmployeeArray(Employee *array, int n)
{
  for (int i=0; i<n; i++)
     {free(array[i].first);
      free(array[i].last);
     }
  free(array);}
```

3. Write two differences between malloc and calloc.

The Calloc takes two arguments, malloc takes one. Malloc does not initialize the memory allocates but Calloc does with zero.

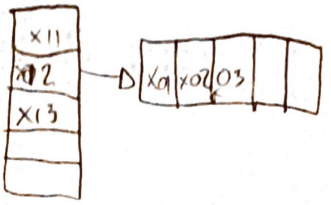
4. What is the purpose of realloc?

The Purpose of realloc is to resize a block of memory that was Previously allocated, It can make it larger or smaller

5. Write few lines of code that creates two arrays with malloc. Then write a statement that can create a memory leak. Discuss why you think your code has a memory leak by drawing the status of the memory after you use malloc and the line of the code you claim that creates a memory leak.

```
int m=5
int * arrays = (int * ) malloc(4 * sizeof(m));
for(int i=0; i<m; i++)
  { int *arrays = (int *) malloc (size of(int));
  }
}
```

the memory is not freed



the memory leak can happen because we have to free the memory for both arrays

6. This question relies of the following structure definition.

```
struct Student {
        int student_id;
        float *quizzes;
}
```

Complete the following function that takes 2 int indicating number of students N and number of quizzes Q. The function allocate appropriate memory for N Students and for each student it should allocate memory for Q quizzes. Then take input for all the data and return the pointer.

```
struct Student* AllocateStudents( int N, int Q) {
struct Student * Students = (struct Student) malloc (N * size of (struct Student));
for(int i=0; i<N; i++)
  {
      for(int j=0; j<Q; j++)
        {
          Students[i].quizes = malloc(size of(float));
        }
  }
return Courses;
}
```

Consider you are calling the function by the following statement:

```
struct Student *students = AllocateStudents(5, 2);
free_up_memory (students, 2);
//write this function on the right side to free up the memory
```

```
free_up_memory(struct Student * st, int n)
 {
  for(int i=0; i<n; i++)
    {
  free(st[i].quizes);
    }
free(st); }
```

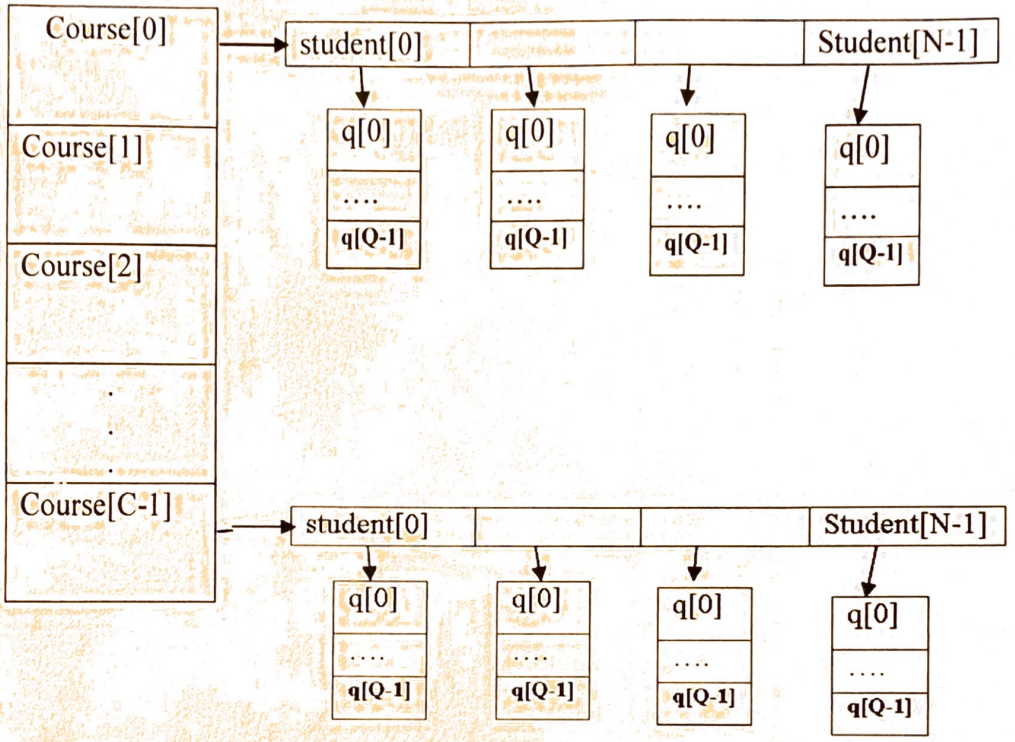
7.

This question relies of the following structure definition.

```
typedef struct Student {
        int student_id;
        float *quizzes;
} Student;
```

There are C number of courses and each course has N number of students and each student has to take Q number of quizzes.

You can visualize it by the following picture.



a) Write a function that takes C, N, and Q and returns appropriate memory to store the data. //You don't have to fill-up the memory with any data. Just allocate the memory.

```
Student **  AllocateCourse_Students(int C, int N, int Q){
{
    Student **Courses = (struct Student **) malloc (C * sizeof(struct Student*));
    for(int i=0; i<N; i++)
    {
        Courses[i] = (struct student*) malloc (N * sizeof(struct student));
        for(int J=0; J<Q; J++)
            Courses[i].quizes = malloc (Q * sizeof(float));
    }
}
```

b) Write a function to free-up the memory. Provide appropriate parameter for the function.

```
Int Main (Void)
{freeMemory( student , C, N, Q)

return 0;
}


Void freeMemory (Student **st, int C, int N, int Q)
 {
   for(int i=0; i<C ; i++ )
     {
     for (int J=0; J<N; J++)
       {
         free(C.st[i].quizzes);
     }
   free(st[i]);
 }free(st
}
```