Lab #4

Spring 2025

Requirements

In some sense, Lab #4 is a repeat of the concepts you used for Lab #3, but . . . no file I/O. Your job is to implement 4 functions detailed in the header file for working with jagged smart arrays. In addition, you are to implement 6 functions for working with a particular ADT (the ADT is a "Course" like a college course). Here is the info from the header file:

```
// General Prototypes for 2D Jagged Arrays
void ** allocateJagged2DArray(int elementSize, int rows, int lengths[]);
void freeJagged2DArray(void ** arrayJagged);
int getRowCount(void ** arrayJagged);
int getColCount(void ** arrayJagged, int row);

// Specific ADT (Course) to test with Jagged Arrays
struct _Course
{
    // The first 3 fields form the course "info"
    int number;
    char department[64];
    char name[64];
};
typedef struct _Course Course;

// Specific functions for the Course ADT

// construcor - return null on a fail
Course * createCourse(int number, char * department, char * name);

// destructor
void freeCourse(Course * pCourse);

// getters - return 1 for success or 0 for failure
int getCourseInfo(Course * pCourse, char * outputString);

// setters - return 1 for success or 0 for failure
int setCourseNumber(Course * pCouurse, int number);
int setCourseDepartment(Course * pCourse, char * department);
int setCourseName(Course * pCourse, char * name);
```

That might seem like a lot of functions, but many of them are very simple. My entire implementation of lab4.c is only 167 lines including a lot of whitespace and comments.

My tests use the jagged smart arrays for integers, Courses, and for another type defined only in main(). To make sure you don't have memory leaks, you may want to try running your code with valgrind.

Sample Output

Here is an example of running my implementation with valgrind:

```
[jer676@hellbender-login lab4]$ valgrind ./a.out
==1583031== Memcheck, a memory error detector
==1583031== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1583031== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1583031== Command: ./a.out
==1583031==
*** Here comes test set #1 - just integers in jagged array ***
RowCount Test: expected 4 was 4
ColCount: expected 2 was 2
ColCount: expected 4 was 4
ColCount: expected 3 was 3
ColCount: expected 7 was 7
array[0][0]=0, array[0][1]=1,
array[1][0]=2, array[1][1]=3, array[1][2]=4, array[1][3]=5,
array[2][0]=6, array[2][1]=7, array[2][2]=8,
array[3][0]=9, array[3][1]=10, array[3][2]=11, array[3][3]=12, array[3][4]=13, array[3][5]=14, array[3][6]=15,



*** Here comes test set #2 - the Course ADT ***
CMP_SC 1050 - Algorithm Design and Programming I
CMP_SC 2050 - Algorithm Design and Programming II
CMP_SC 3050 - Advanced Algorithm Design
CMP_SC 4520 - Operating Systems I




*** Here comes test #3 - some data type the user used, but you've never seen ***
Jim Ries (Human) - 57, Laura Ries (Human) - 55, Abbie Ries (Human) - 25, Charlotte Ries (Human) - 19, Cisco Ries (Canine
) - 13, Murphy Ries (Canine) - 1,
Larry Ries (Human) - 55, Allison Ries (Human) - 14,

[jer676@hellbender-login lab4]$

*** Test set #4: Try to blow things up ***
Try to blow up allocateJagged2DArray()
Try to blow up freeJagged2DArray()
Try to blow up createCourse()
Try to blow up freeCourse()
Try to blow up getCourseInfo()
Try to blow up setters


DONE!
==1583031==
==1583031== HEAP SUMMARY:
==1583031==     in use at exit: 0 bytes in 0 blocks
==1583031==   total heap usage: 15 allocs, 15 frees, 8,416 bytes allocated
==1583031==
==1583031== All heap blocks were freed -- no leaks are possible
==1583031==
==1583031== For lists of detected and suppressed errors, rerun with: -s
==1583031== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[jer676@hellbender-login lab4]$
```

Note that my first test is a jagged array of type int with 4 rows and column sizes 2,4,3,7. The second test is a jagged array of type FamilyMember (a struct I defined in my main file) with 2 rows and column sizes 6,2.

Hints

If you are confident with jagged arrays, do that first. If not, you can get some easy points by working on the Course ADT functions which allow you to create a Course, free a Course, and get and set various things. You should write your own main() with your own tests. When you have something working, go ahead and submit it using mucsmake even if you have only down a small part of the assignment. You can keep adding functions until they all work (or, worst case, you run out of time).

Submission Information

Submit this assignment by using the mucsmake command.

Use the following command on Hellbender:

mucsmake <course> <assignment> <filename>

For example:

mucsmake 2050 lab4 lab4.c

Rubric: 20 points

1. allocateJagged2DArray() – 3 points
2. freeJagged2DArray() – 3 points
3. getRowCount() – 3 points
4. getColCount() – 3 points
5. createCourse – 2 points
6. freeCourse – 1 point
7. getCourseInfo – 2 points
8. setCourseNumber – 1 point
9. setCourseDepartment – 1 point
10. setCourseName – 1 point

Notice:

1. All of your lab submissions **must** include documentation to receive full points.
2. All of your lab submissions must compile under GCC using the *−Wall* and *−Werror* flags to be considered for a grade.
3. You are expected to provide proper documentation in every lab submission, in the form of code comments. For an example of proper lab documentation and a clear description of our expectations, see the lab policy document.