Aaron Jones
CSCD 240 Lab 8

CSCD 240

NOTE: Your answers, for all problems, will be saved in a file named cscd240Lab8.pdf for all problems
NOTE: Your C file will be named cscd240Lab8.c

1) Type in, compile and execute the following code.

```
#include <stdio.h>
#include <stdlib.h>
#define R 5
#define C 4

int main(int argc, const char * argv[])
{

   int i, j, walk = 0;
   int **grades = (int **)malloc(R * sizeof(int *));  //Assume that value of this entire expression is 0x7fb322403930

   // allocate memory for each row for test purpose, otherwise we should do the following in a loop
   grades[0] = (int *) malloc( C * sizeof(int) );       //Assume that value of this entire expression is 0x7fb3224000e0
   grades[1] = (int *) malloc( C * sizeof(int) );       //Assume that value of this entire expression is 0x7fb322401f50
   grades[2] = (int *) malloc( C * sizeof(int) );
   grades[3] = (int *) malloc( C * sizeof(int) );
   grades[4] = (int *) malloc( C * sizeof(int) );

   for( i = 0; i < R; i ++)                    //This will navigate the array by the rows
   {
      for(j = 0; j < C; j ++)                  //This will navigate the array by the columns next
      {
         grades[i][j] = walk ++;               //The values of grades[i][j] will get filled with the values of walk as it
                                               //increments after it navigates the whole array. Filling it with the numbers
      }                                        // 0 – 19 from left to right.
   }

   int **pptr = grades;                        //this double pointer will point to another double pointer grades as well as
                                               //getting the values of grades.
   printf("-1: pptr= %p\n", pptr);             /* The pptr in this example will just return the memory location of pptr.
   printf("-1: &pptr[0] = %p\n", &pptr[0] );   *  The next print statement will pretty much return the same memory
   printf("-1: pptr+1= %p\n", pptr + 1);       *  location as the first print statement. Then the third and forth will
   printf("-1: pptr+2= %p\n", pptr + 2);       *  move up 8 bytes in memory location because we are manipulating the
                                               *  address memory and each memory address gets 8 bytes.*/
```

Aaron Jones
CSCD 240 Lab 8

```
    printf("\n\n");

    printf("0: pptr[0]= %p\n", pptr[0]); /*This will give us a value of grades[0] as it's return.
    printf("0: *pptr= %p\n", *pptr);       //0x7fb3224000e0 as well the rest of the values as they all point to
    printf("0: &pptr[0][0]= %p\n", &pptr[0][0]); // So the value will be  grades[0].*/

    printf("\n\n");

    printf("1: pptr[1]= %p\n", pptr[1]);                /* These values will give us the value of grade[2] as pptr[1] points to
    printf("1: *(pptr + 1)= %p\n", *(pptr + 1));        * the memory location of grade[2], as do the rest of the pptr's in these
    printf("1: &pptr[1][0] = %p\n", &pptr[1][0] );      * print statements. So it will return 0x7fb322401f50 */

    printf("\n\n");

    printf("2: *pptr + 1 = %p\n", *pptr + 1);           /* The values of these print statements will give us the value +4 bytes
    printf("2: *(pptr+0) + 1 = %p\n", *(pptr + 0) + 1); * up on the memory allocation because we are only adding 1 int to the
    printf("2: &pptr[0][1] = %p\n", &pptr[0][1] );      * pptr. Resulting in the first two being  0x7fb3224000e4 and the last
                                                        * should return a value of 0x7fb322401f50 as it is pointing to grade[2]
                                                        * value*/

    printf("\n\n");
    printf("3: *(pptr[1] + 1) = %d\n", *(pptr[1] + 1) );            /* These print statements should return the actual integer value
    printf("3: *( *(pptr + 1) + 1) = %d\n", *( * (pptr + 1) + 1) ); *  of pptr[1][1]. Which should be equivalent to 5 per the loop
    printf("3: pptr[1][1] = %d\n", pptr[1][1] );                    *  given above filling the value with the value 5.  Also pptr[1]
                                                                    *  [1] is pointing to the value of grades[1][1]. Which is 5.
                                                                    **/

    return 0;
} //end of main
```

This code will provide a base address for grades and pptr.

Assume you get a 64-bit machine, i.e. an address takes 8 bytes long. Also, each integer on your machine takes 4 bytes long. Use the base address provided from the comments to the right of malloc() function. Based on the code above, create an educated guess that clearly outlines what you believe will print out as each line of code above is executed. In your explanation clearly **explain what is happening**, don't just give memory addresses or values. If you only provide memory addresses or values you will receive **0 points** for this problem. Your guesses will be clearly labeled in the PDF file. You must provide the line of code and then the explanation.

Aaron Jones
CSCD 240 Lab 8

TO TURN IN:
A zip file containing:

- Your PDF file
- Your C file

You better know the naming scheme.