```c
/* This program shows examples of 2 different ways of declaring, and
 * accessing 2D arrays and of how they are arranged in memory
 * (1) statically allocated: (all buckets of the array are stored in
 *     row-major order, and all buckets of the array are contiguous)
 * (2) dynamically allocated in a single malloc: (all buckets of the
 *     array are contiguous)
 */

#include <stdio.h>
#include <stdlib.h>

#define N  3
#define M  4

void init_static(                ); // TODO
void init_dynamic(               ); // TODO

int sum_static(                  ); // TODO
int sum_dynamic(                 ); // TODO

int main() {
    // a statically allocated 2D array with N rows and M columns
    int stat[N][M];

    // a dynamically allocated "2D" array using 1 malloc
    int *dyn;

    int stat_sum, dyn_sum;

    // a dynamically allocated 2D array as a single malloc
    dyn = malloc(sizeof(int)*N*M);  // NxM int values
    if (!dyn) {
     printf("malloc failed\n");
     exit(1);
    }
    // let's initialize these arrays, we need separate functions for
    // each type of array:
    init_static(stat,N,M); // TODO: complete this function
    printf("\n\nStatic 2D array:\n");
    // let's see if our init function worked, print out stat
    for(int i=0;i<N;i++){       //access rows first.
        for(int j=0;j<M;j++){   //within each row, access a column
            printf(         ); //TODO: print each value of stat
        }
    }

    init_dynamic(dyn,N,M); // TODO: complete this function
    printf("\n\nDynamic 2D array with 1 big malloc:\n");
    // let's see if our init function worked, print out dyn
    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            printf(         ); //TODO: print each value of dyn
        }
    }

    // print the return value of sum_static
    stat_sum = sum_static(); // TODO: complete this function

    printf("\n sum of static array: %d",  stat_sum);

    // print the return value of sum_static
    dyn_sum = sum_dynamic(); // TODO: complete this function

    printf("\n sum of static array: %d",  dyn_sum);

    //TODO: free space for dyn: in one call to free and set the
    //      pointer to NULL


    return 0;
}
```

```
/*
 * initialize a statically declared array: item at (i,j) = i+j
 *   array: the array to init, M is the column dimension
 *          the row dimension does not need to be specified here
 *   rows: the number of rows in this array
 *
 * We need to specify the column index to the array parameter
 * so that the compiler can  generate code that will compute where
 * the start of each row is in memory.
 * Passing in rows as a second parameter just makes this more
 * generic than specifying both the exact row and column dimensions
 * in the parameter.
 */
//TODO: complete the function definition and initialize the array
//      as specified above.
void init_static(                          ) {












}
```

```c
/*  initialize a dynamically declared array 2D array
 *  item at (i,j) = i*j
 *  (1 big malloc: an NxM contiguous chunk of ints)
 *   array: the array to init
 *   rows: the number of rows
 *   cols: the number of cols
 *
 *   we CANNOT use [i][j] syntax in here to access elements because
 *   the compiler cannot generate the right code to find the start
 *   of the next row (it depends on the col dimension).  As a result
 *   we will do the calculation in this code to get the right values.
 */
//TODO: complete the function definition and initialize the array
//      as specified above.
void init_dynamic(                      ) {




















}
```

```c
/*
 * return the sum of all elements in the passed
 * statically declared array
 *   array: the array of ints
 *   rows: the number of rows in the passed array
 */
//TODO: complete the function definition and compute the sum
//      and return the sum as specified above.
int sum_static() {












}

/*
 * return the sum of all elements in the passed
 * dynamically declared 2D array (1 big malloc)
 *   array: the array of ints
 *   rows: the number of rows in the passed array
 *   cols: the number of rows in the passed array
 */
int sum_dynamic(                   ) {












}
```

Q1. Consider the following partial program:

```
#include <stdio.h>

struct person {
  char name[32];
  int  age;
  float heart_rate;


};
//TODO: complete these function prototypes
void init_struct_data (                );

void print_struct_data (                  );

int main(void) {
  struct person  p1;
  struct person patients[2];
  init_struct_data(&p1);
  print_struct_data(  ); //TODO: declare a prototype and function definition.
  return 0;
}

/* This function takes in a pointer to a struct person,
 * and initializes all the fields of the person. You can
 * set arbitrary values for the fields of the struct.
 */
//TODO: complete the function definition
void init_struct_data(     ){








}
```

```
/* This function takes in a struct person,
 * and prints all the fields of the struct.
 */
//TODO: complete the function definition
void print_struct_data(                ){





}
```

(1) What type is each of the following expressions?

expression	type
p1	
p1.name	
p1.heart_rate	
patients	
patients[0]	
patients[0].name	
patients[0].name[3]	

(2) Write the C code to set the age of the 2nd person in the patients array to 18:

(3) Let's say we wanted to keep track of each person's height and weight. What changes would we need to make to our code? Show your changes inline in the code above.