Lab 3

Task 1: Array Storage

The Functions and data types

We implemented the following function:

- `char *two_d_alloc(int M, int N, size_t size)`
  - Allocates an M * N 2D array with space for *size* entries.

- `void two_d_dealloc(char *array)`
  - Deallocates a matrix located at ***array*.

- `void two_d_fetch(char *start, size_t size_of_element, int matrix_size_C, int matrix_size_R, int R, int C, void *result)`
  - Fetches element of size *size_of_element* from a *matrix_size_R* * *matrix_size_C* matrix at index *R* * *C* at the memory address *start* and returns it to address *result*.

- void two_d_store(char *start, size_t size, int matrix_C, int matrix_R, int R, int C, void *value)
  - Store element of size *size* from a *matrix_R* * *matrix_C* matrix at index *R* * *C* at the memory address *start* with value *value*.

- void two_d_fetch_column(char *start, size_t size_of_element, int matrix_size_C, int matrix_size_R, int R, int C, void *result)
  - Same as two_d_fetch only column wise logic.

- void two_d_store_column(char *start, size_t size, int matrix_C, int matrix_R, int R, int C, void *value)
  - Same as two_d_store only column wise logic.

- void two_d_fetch_pointer(char *start, size_t size_of_element, int matrix_size_C, int matrix_size_R, int R, int C, int *result)
  - Same as two_d_fetch but it fetches an integer pointer

- void two_d_store_pointer(char *start, size_t size, int matrix_C, int matrix_R, int R, int C, int *value)
  - Same as two_d_store only it stores an integer pointer.

We also implanted a struct to use our arbitrary array.

- Struct Box
  - Data:
    - Int a;
    - Int b;
    - Int c;

The Tests and Result

This is how we ran our functions and preformed all the tests that was required:

```c
int main()
{

    char *d = two_d_alloc(10, 20, sizeof(int));
    int new_value = 1337;
    two_d_store(d, sizeof(int), 10, 20, 1, 2, &new_value);
    int fetch;

    two_d_fetch(d, sizeof(int), 10, 20, 1, 2, &fetch);
    printf("Int fetch: %d\n", fetch);
    two_d_fetch(d, sizeof(int), 10, 20, 10000, 100000, &fetch);

    two_d_dealloc(d);

    char *pointers = two_d_alloc(5, 10, sizeof(int*));
    int test = 123;
    int *test_pointer;
    test_pointer = &test;
    two_d_store_pointer(pointers, sizeof(int*), 5, 10, 2, 3, test_pointer);
    int *pointer_fetch = malloc(sizeof(int*));
    two_d_fetch_pointer(pointers, sizeof(int*), 5, 10, 2, 3, pointer_fetch);
    printf("pointer fetch = %d\n", *pointer_fetch);
    two_d_dealloc(pointers);

    struct Box box;
    box.a = 100;

    char *struct_array = two_d_alloc(4, 2, sizeof(struct Box));
    two_d_store(struct_array, sizeof(struct Box), 4, 2, 1, 1, &box);
    struct Box box_fetch;
    two_d_fetch(struct_array, sizeof(struct Box), 4, 2, 1, 1, &box_fetch);

    printf("Box data: %d", box_fetch.a);
    two_d_dealloc(struct_array);
```

Code 1

This is the result from running that code:

```
Int fetch: 1337
Out of index request!
pointer fetch = 123
Box data: 100

------------------
(program exited with code: 0)
Press return to continue
```

Code 2

Task 2: Memory Dump

We implemented a HEX memory dumper.

The functions and Code

We use the code from the previous exercise to create an array for the characters in the string we defined in the code. We also made an array to store all the hex representation of the string which was later converted back to ascii according to the exercise specification.

We also made a function to convert a hex-number to ascii format character.

```
char hexToAscii(char first, char second)
{
char hex[2], *stop;
hex[0] = first;
hex[1] = second;
return strtol(hex, &stop, 16);
}
```

Here is the main code:

```
int main()
{
    char string[] = "Hello my name is JEFF and im a huge fan of hotdogs in the night";
    int len = sizeof(string);
    char* array = two_d_alloc(len, 1, sizeof(char));
    for (int i = 1; i<len; i++)
    {
        two_d_store(array,sizeof(char), len, 1, 1, i, &string[i-1]);
    }
    char test;
    two_d_fetch(array, sizeof(char), len, 1, 1, len-1, &test);
    printf("Test = %c\n", test);

    char* hexarray = two_d_alloc(len, 1, sizeof(char)*2);


    printf("00 ");
    for (int i = 1; i <= len; i++)
    {

        char *temp = malloc(sizeof(char)*2);
        char fetch;
        two_d_fetch(array, sizeof(char), len, 1, 1, i, &fetch);
        sprintf(temp, "%x", fetch);
        printf("%s", temp);
        two_d_store(hexarray, sizeof(char)*2, len, 1, 1, i, temp);

        if(i%4 == 0 && i != 0)
        {

            if(i%16 == 0) //End of line
            {
                printf(" ");
                for (int j = 0; j<16; j++)
                {
                    if(j%4 == 0 && j != 0)
                    {
                        printf(" ");
                    }
                    char *hex = malloc(sizeof(char)*2);
                    two_d_fetch(hexarray, sizeof(char)*2, len, 1, 1, i+j-16+1, hex);

                    printf("%c", hexToAscii(hex[0], hex[1]));
                }

                printf("\n%x ", i); //New line number

            }
            else
            {
                printf(" ");
            }
        }
    }
    printf("\n");
}
```

Here is the result from running the code: