Computer Lab 3

Memory Organization

Computer Architectures for MSc in Engineering

HT19, DT509G

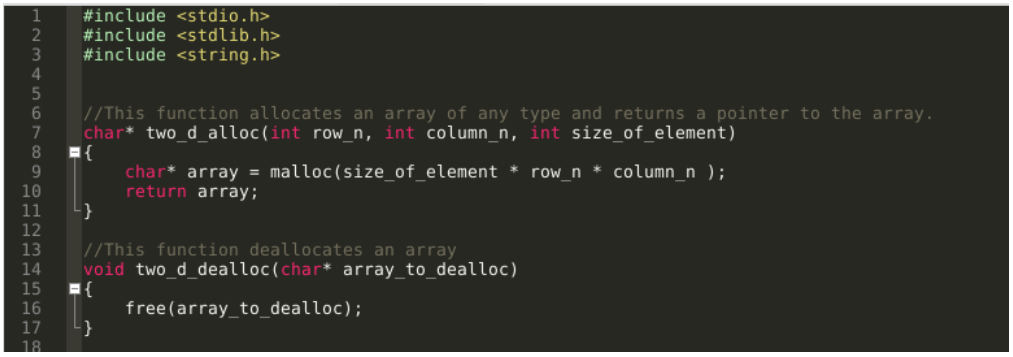
Abshir Abdulrahman & Fria Khorshid

2019-10-10

Task 1: Array Storage

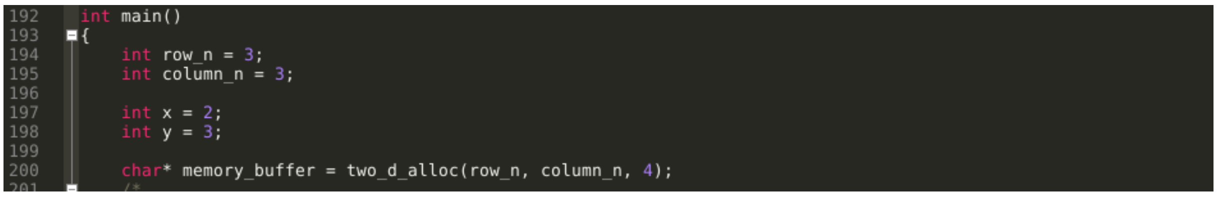
The main idea of this task is to create a two-dimensional byte array that use linear storage in c. We are going to implement two functions, two_d_store to store a value of into the 2D array and two_d_fetch to access value in the 2D array.

We are allocating a char* called array of size row_n * column_n * size of each element in the two_d_alloc function. In the second function, two-d_dealloc we are deallocating the created two-dimensional array.

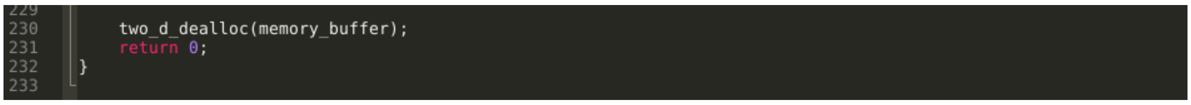


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//This function allocates an array of any type and returns a pointer to the array.
char* two_d_alloc(int row_n, int column_n, int size_of_element)
{
    char* array = malloc(size_of_element * row_n * column_n );
    return array;
}

//This function deallocates an array
void two_d_dealloc(char* array_to_dealloc)
{
    free(array_to_dealloc);
}
```

In the main function we are allocating an array(memory_buffer) of 3*3, which the size of each element is 4 and then we deallocating the same array right after to test both function works.

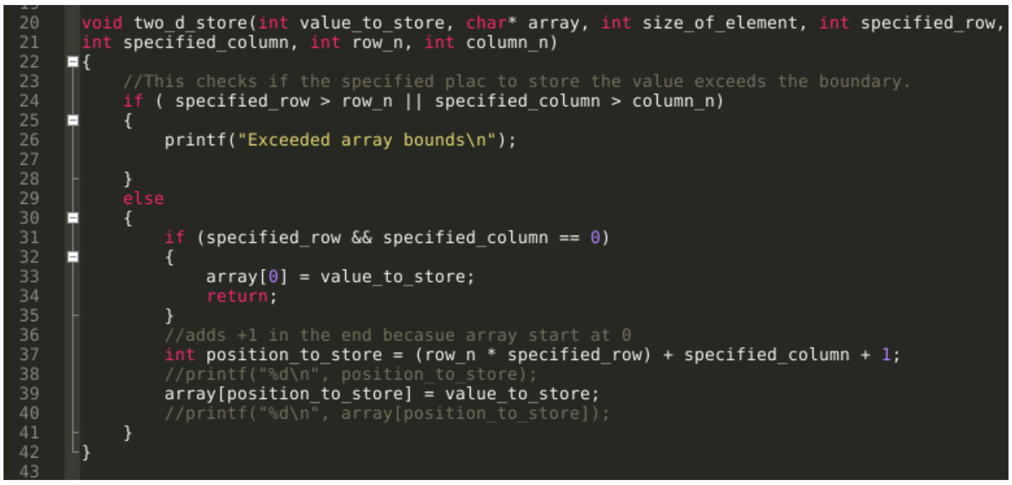


```c
int main()
{
    int row_n = 3;
    int column_n = 3;

    int x = 2;
    int y = 3;

    char* memory_buffer = two_d_alloc(row_n, column_n, 4);
```



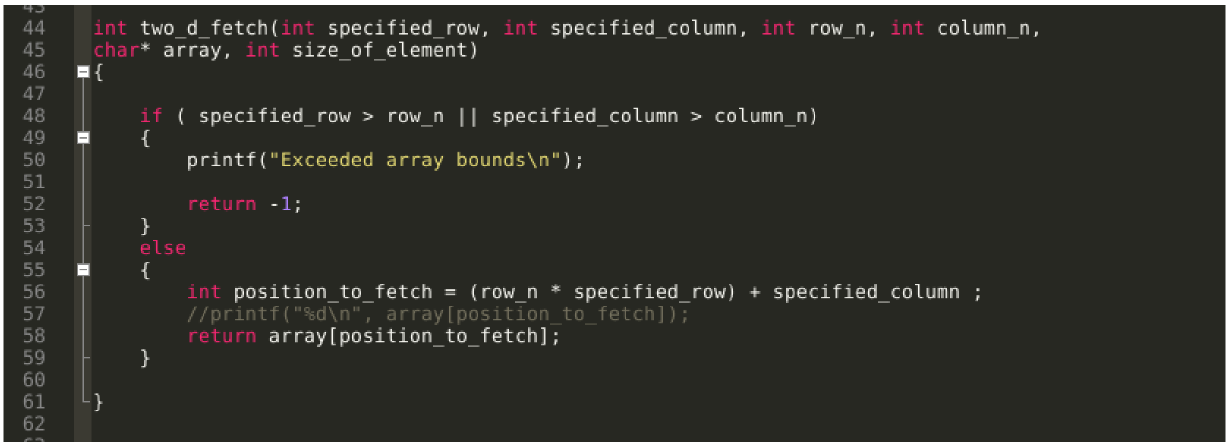
```c
    two_d_dealloc(memory_buffer);
    return 0;
}
```

Here we are creating a function, two_d_store to store a value in the created array at specific position using the row-major form. The function takes in as argument, value_to_store(the specific integer you want to store in the array), array(which is the created two-dimensionel array), size_of_element(the size of each element), specific_row and specific_column(the position at which you want to store in the array), row_n and column_n (which specifies the size of the array). The function checks first boundary condition, if the position you want to store the integer in the array exceeds the bounds for the array. Otherwise the integern will be stored in the array at the position based on the row-major form.



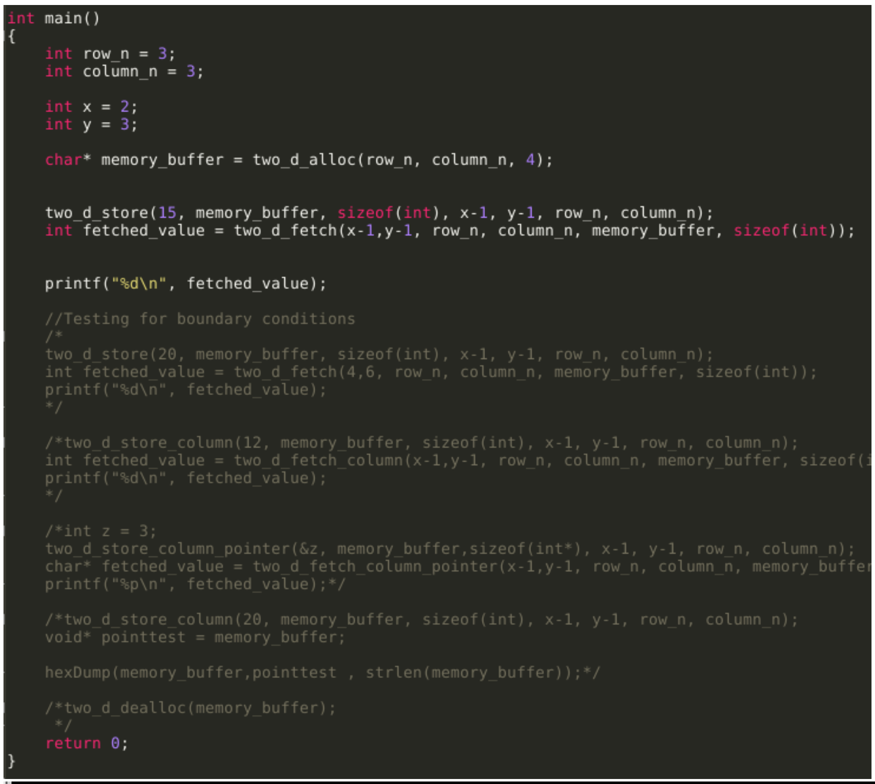
```c
void two_d_store(int value_to_store, char* array, int size_of_element, int specified_row,
int specified_column, int row_n, int column_n)
{
    //This checks if the specified plac to store the value exceeds the boundary.
    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

    }
    else
    {
        if (specified_row && specified_column == 0)
        {
            array[0] = value_to_store;
            return;
        }
        //adds +1 in the end becasue array start at 0
        int position_to_store = (row_n * specified_row) + specified_column + 1;
        //printf("%d\n", position_to_store);
        array[position_to_store] = value_to_store;
        //printf("%d\n", array[position_to_store]);
    }
}
```

In the second function, which has the same input arguments as two_d_store function except the value_to_store, which defines the specific integer you want to store in the array, is used to return a value from the array . Here the the same boundary condition is used to check if we are trying to access a value that is not in the array. If that's not the case return the value from specfic position in the array.

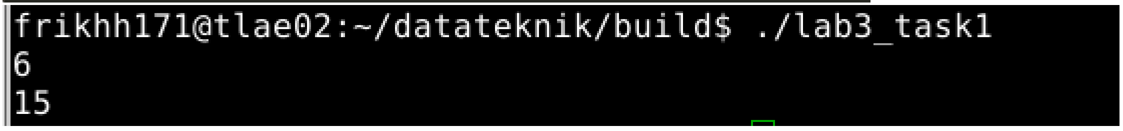


```c
int two_d_fetch(int specified_row, int specified_column, int row_n, int column_n,
char* array, int size_of_element)
{

    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

        return -1;
    }
    else
    {
        int position_to_fetch = (row_n * specified_row) + specified_column ;
        //printf("%d\n", array[position_to_fetch]);
        return array[position_to_fetch];
    }

}
```

Here we are first creating an array of size 3*3, then we are trying to store the value 15 in the position (2,3) of a two-dimensionel array, which will be position 6, in row-major form. Then we are trying to fetch the same value in the array and print it.

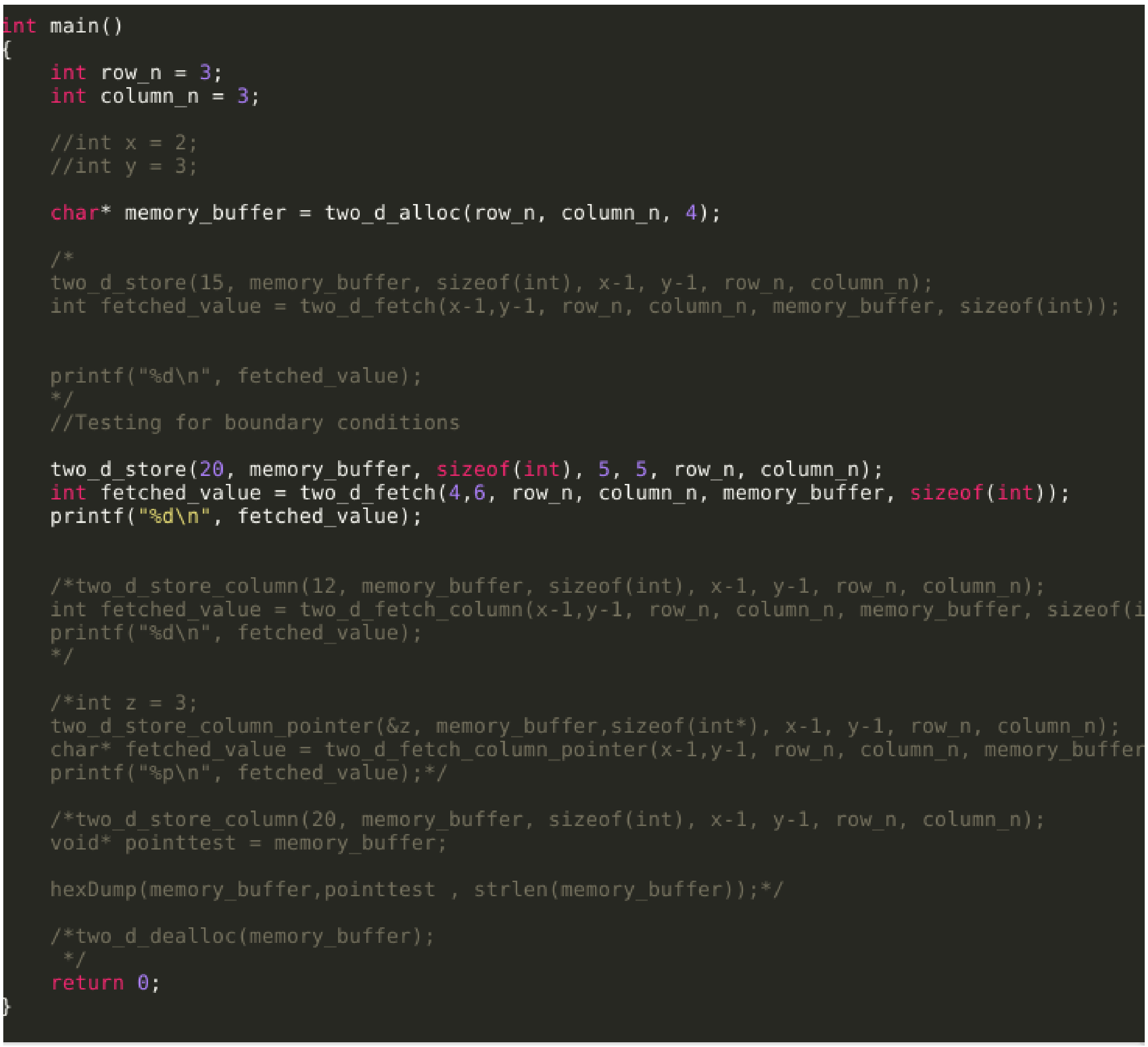


```c
int main()
{
    int row_n = 3;
    int column_n = 3;

    int x = 2;
    int y = 3;

    char* memory_buffer = two_d_alloc(row_n, column_n, 4);


    two_d_store(15, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));


    printf("%d\n", fetched_value);

    //Testing for boundary conditions
    /*
    two_d_store(20, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch(4,6, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);
    */

    /*two_d_store_column(12, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch_column(x-1,y-1, row_n, column_n, memory_buffer, sizeof(i
    printf("%d\n", fetched_value);
    */

    /*int z = 3;
    two_d_store_column_pointer(&z, memory_buffer,sizeof(int*), x-1, y-1, row_n, column_n);
    char* fetched_value = two_d_fetch_column_pointer(x-1,y-1, row_n, column_n, memory_buffer
    printf("%p\n", fetched_value);*/

    /*two_d_store_column(20, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    void* pointtest = memory_buffer;

    hexDump(memory_buffer,pointtest , strlen(memory_buffer));*/

    /*two_d_dealloc(memory_buffer);
     */
    return 0;
}
```

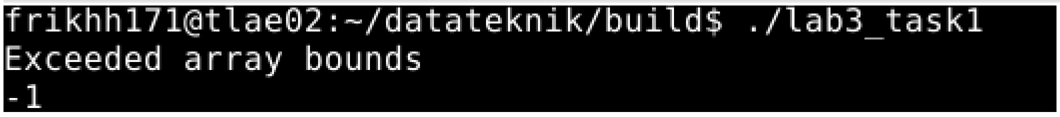


```
frikhh171@tlae02:~/datateknik/build$ ./lab3_task1
6
15
```

This is for testing the boundary condition. We are trying to store value 20 at position (5,5), which is not a position in the 3*3 array created. That’s why the output Exceed array bounds is printed. Then we are trying to fetch a value at position (4,6) in the array, which is resulted in -1 because of we are trying to fetch a value at a position that’s out of bounds, not in our array.

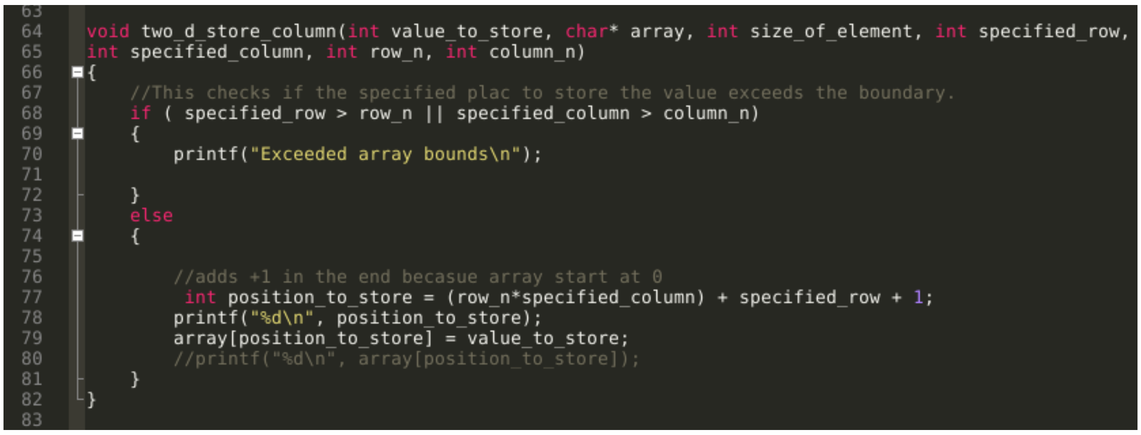


```
int main()
{
    int row_n = 3;
    int column_n = 3;

    //int x = 2;
    //int y = 3;

    char* memory_buffer = two_d_alloc(row_n, column_n, 4);

    /*
    two_d_store(15, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));


    printf("%d\n", fetched_value);
    */
    //Testing for boundary conditions

    two_d_store(20, memory_buffer, sizeof(int), 5, 5, row_n, column_n);
    int fetched_value = two_d_fetch(4,6, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);


    /*two_d_store_column(12, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch_column(x-1,y-1, row_n, column_n, memory_buffer, sizeof(i
    printf("%d\n", fetched_value);
    */

    /*int z = 3;
    two_d_store_column_pointer(&z, memory_buffer,sizeof(int*), x-1, y-1, row_n, column_n);
    char* fetched_value = two_d_fetch_column_pointer(x-1,y-1, row_n, column_n, memory_buffer
    printf("%p\n", fetched_value);*/

    /*two_d_store_column(20, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    void* pointtest = memory_buffer;

    hexDump(memory_buffer,pointtest , strlen(memory_buffer));*/

    /*two_d_dealloc(memory_buffer);
     */
    return 0;
}
```

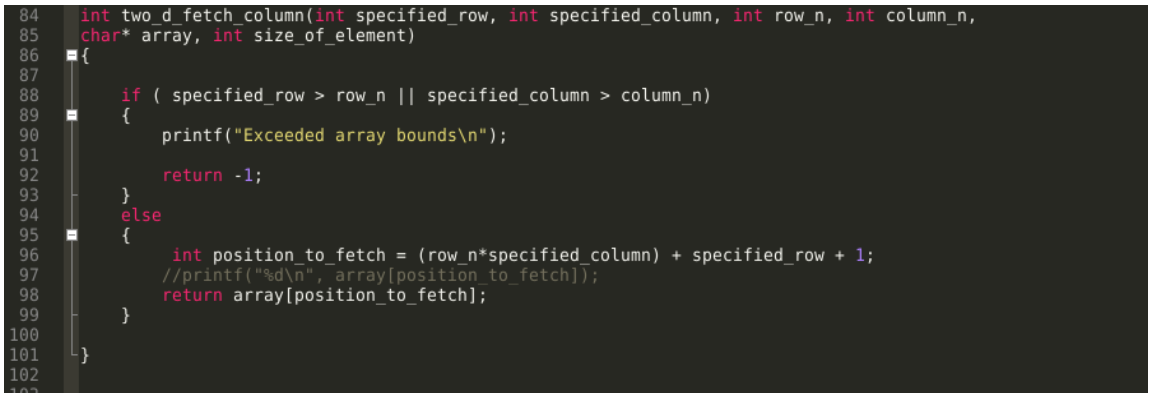


```
frikhh171@tlae02:~/datateknik/build$ ./lab3_task1
Exceeded array bounds
-1
```

Here we are implementing two functions, two_d_store_column and two_d_fetch_column that's very similar to the two earlier functions, two_d_store and two_d_fetch except that we are rewriting the formula for row-major form to column-major form, simply by taking the column at which the value is to be stored times the size of the row of the array plus the row at which the value is to be stored and 1.

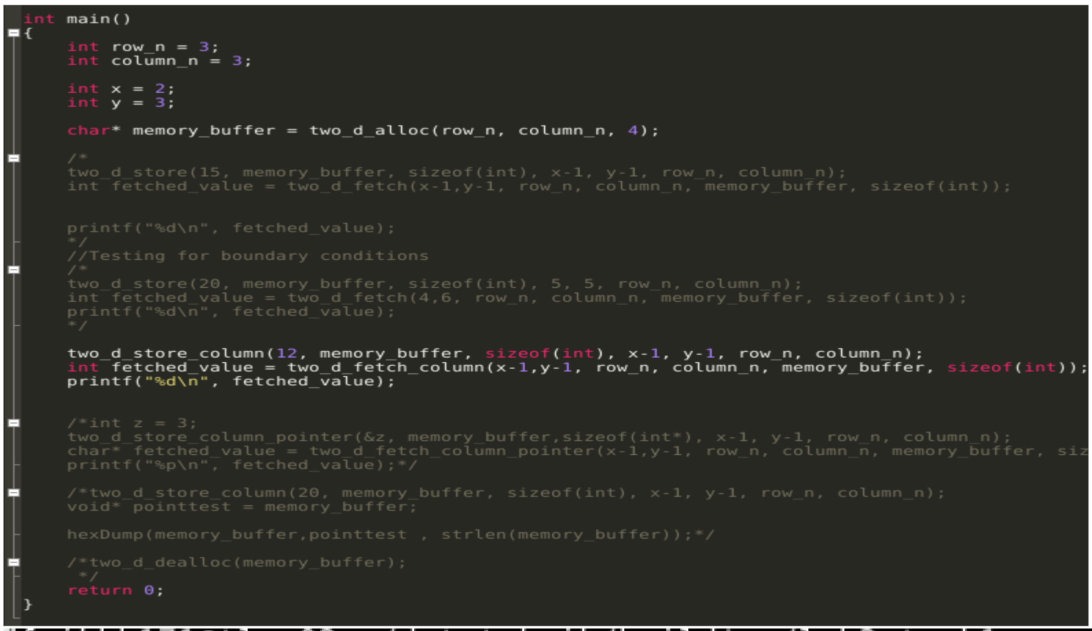


```c
void two_d_store_column(int value_to_store, char* array, int size_of_element, int specified_row,
int specified_column, int row_n, int column_n)
{
    //This checks if the specified plac to store the value exceeds the boundary.
    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

    }
    else
    {

        //adds +1 in the end becasue array start at 0
         int position_to_store = (row_n*specified_column) + specified_row + 1;
        printf("%d\n", position_to_store);
        array[position_to_store] = value_to_store;
        //printf("%d\n", array[position_to_store]);
    }
}
```

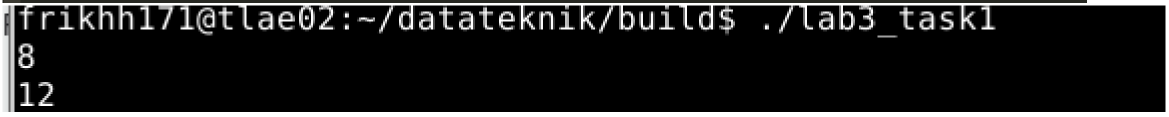


```c
int two_d_fetch_column(int specified_row, int specified_column, int row_n, int column_n,
char* array, int size_of_element)
{

    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

        return -1;
    }
    else
    {
         int position_to_fetch = (row_n*specified_column) + specified_row + 1;
        //printf("%d\n", array[position_to_fetch]);
        return array[position_to_fetch];
    }

}
```

We are testing these two functions using column-major format, by allocating the value 12 in the position (2,3) in the two-dimensional array and then fetching it. The value 12 should be in position 8 in column-major order.

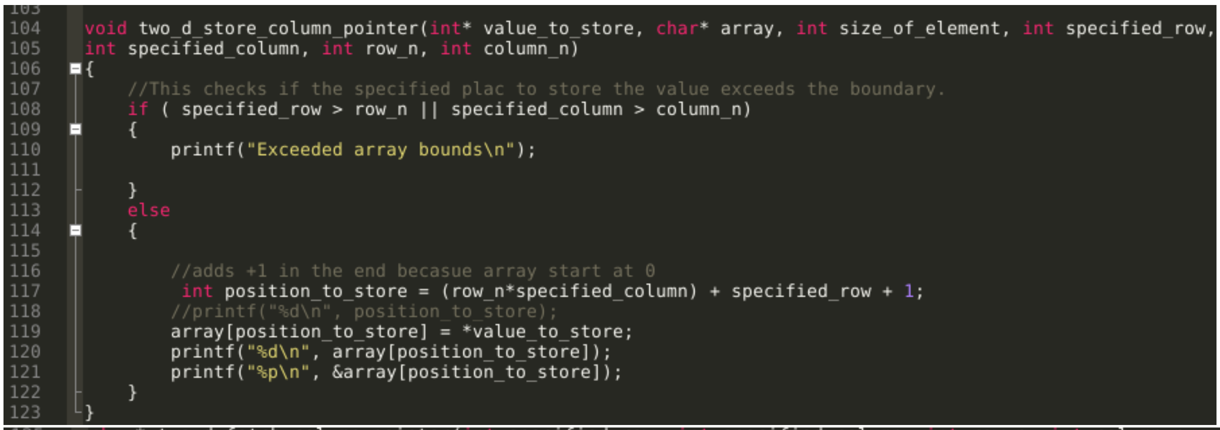


```c
int main()
{
    int row_n = 3;
    int column_n = 3;

    int x = 2;
    int y = 3;

    char* memory_buffer = two_d_alloc(row_n, column_n, 4);

    /*
    two_d_store(15, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));


    printf("%d\n", fetched_value);
    */
    //Testing for boundary conditions
    /*
    two_d_store(20, memory_buffer, sizeof(int), 5, 5, row_n, column_n);
    int fetched_value = two_d_fetch(4,6, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);
    */

    two_d_store_column(12, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch_column(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);


    /*int z = 3;
    two_d_store_column_pointer(&z, memory_buffer,sizeof(int*), x-1, y-1, row_n, column_n);
    char* fetched_value = two_d_fetch_column_pointer(x-1,y-1, row_n, column_n, memory_buffer, siz
    printf("%p\n", fetched_value);*/

    /*two_d_store_column(20, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    void* pointtest = memory_buffer;

    hexDump(memory_buffer,pointtest , strlen(memory_buffer));*/

    /*two_d_dealloc(memory_buffer);
     */
    return 0;
}
```

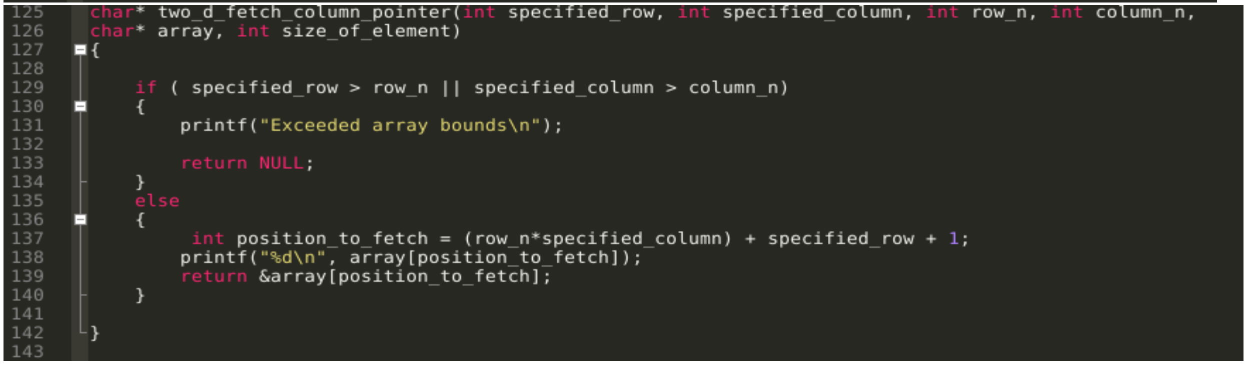


```
frikhh171@tlae02:~/datateknik/build$ ./lab3_task1
8
12
```

Previously functions had an input argument, which accepted integer by value (the value at a specific position), but these two functions, two_d_store_column_pointer and two_d_fetch_column_pointer, are modified to accept input argument of integer pointer. The idea is same as the before, but here we are trying to store a value using a pointer and return the address of the memory where the value is stored. Here we are using column-major form.

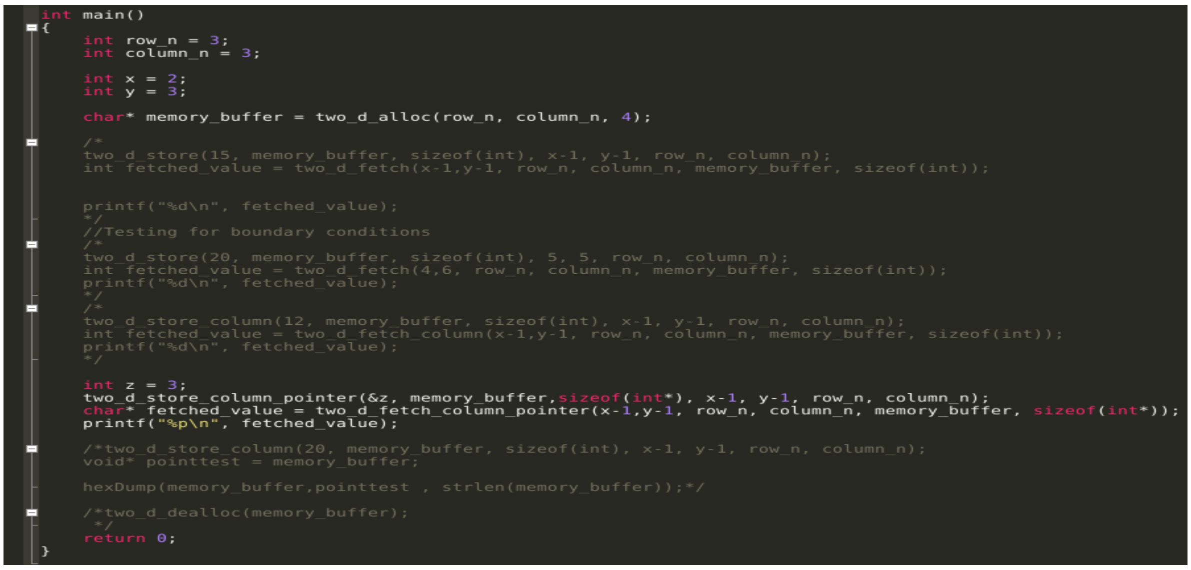


```c
void two_d_store_column_pointer(int* value_to_store, char* array, int size_of_element, int specified_row,
int specified_column, int row_n, int column_n)
{
    //This checks if the specified plac to store the value exceeds the boundary.
    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

    }
    else
    {

        //adds +1 in the end becasue array start at 0
         int position_to_store = (row_n*specified_column) + specified_row + 1;
        //printf("%d\n", position_to_store);
        array[position_to_store] = *value_to_store;
        printf("%d\n", array[position_to_store]);
        printf("%p\n", &array[position_to_store]);
    }
}
```

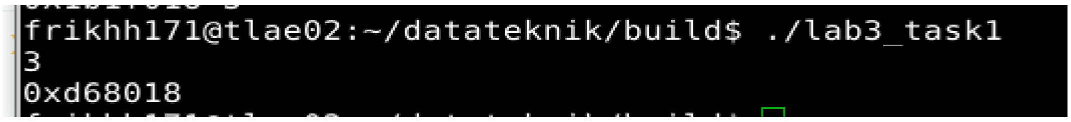


```c
char* two_d_fetch_column_pointer(int specified_row, int specified_column, int row_n, int column_n,
char* array, int size_of_element)
{

    if ( specified_row > row_n || specified_column > column_n)
    {
        printf("Exceeded array bounds\n");

        return NULL;
    }
    else
    {
         int position_to_fetch = (row_n*specified_column) + specified_row + 1;
        printf("%d\n", array[position_to_fetch]);
        return &array[position_to_fetch];
    }

}
```

Here we are inserting the address at which the value 3 is stored in memory in the two_d_store_column_pointer function and then we are fetching the address of that memory location in the other function and printing it below.



```c
int main()
{
    int row_n = 3;
    int column_n = 3;

    int x = 2;
    int y = 3;

    char* memory_buffer = two_d_alloc(row_n, column_n, 4);

    /*
    two_d_store(15, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));


    printf("%d\n", fetched_value);
    */
    //Testing for boundary conditions
    /*
    two_d_store(20, memory_buffer, sizeof(int), 5, 5, row_n, column_n);
    int fetched_value = two_d_fetch(4,6, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);
    */
    /*
    two_d_store_column(12, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    int fetched_value = two_d_fetch_column(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int));
    printf("%d\n", fetched_value);
    */

    int z = 3;
    two_d_store_column_pointer(&z, memory_buffer,sizeof(int*), x-1, y-1, row_n, column_n);
    char* fetched_value = two_d_fetch_column_pointer(x-1,y-1, row_n, column_n, memory_buffer, sizeof(int*));
    printf("%p\n", fetched_value);

    /*two_d_store_column(20, memory_buffer, sizeof(int), x-1, y-1, row_n, column_n);
    void* pointtest = memory_buffer;

    hexDump(memory_buffer,pointtest , strlen(memory_buffer));*/

    /*two_d_dealloc(memory_buffer);
     */
    return 0;
}
```



```
frikhh171@tlae02:~/datateknik/build$ ./lab3_task1
3
0xd68018
```