Lab 3

Isa Ertunga
Hannes Halm

Task 1 (array storage)

What follows is the C code for task 1. The answers will be explained in the same order as the requirements in the task.

- This is done in the two_d_alloc function, where the C function “malloc” is used to allocate memory for the array.
- Two_d_dealloc deallocates using the “free” function in C
- The functions in the main functions are shown below.
- As shown below the two_d_store function is implemented. Which stores an integer into the array in the correct position, using row major order. A later requirement was that column major order were to be implemented too. The code snippet below implements column major order. However row major can be implemented if one “uncomments” the line in the function.
- Two_d_fetch returns the requested value.
- We have implemented conditional statements for boundary conditions.
- The fetch and store functions are written to support both formats, however the program has to choose which format by changing comments in the code.
- The two function two_pd_store and two_pd_fetch implements the same functionality as before. However instead of integers. It uses integer pointers.

```
#include <stdio.h>
#include <stdlib.h>
//int *two_d_array[5][5];



char * two_d_alloc (int n, int m, int size) {
```

```
	char *two_d_array = (char *)malloc(n * m * size);

	return two_d_array;
}

void two_d_dealloc (char *p) {
	free(p);
}

void two_d_store (char *p, int size, int n, int m, int rows, int
columns, int value) {
	if (n < rows && n >= 0 && m < columns && m >= 0) {
		if (size <= sizeof(value)) {
			p[rows * m + n] = value; //column major
			//p[columns * n + m] = value;    //Row major

		}
	}
	else
		printf("out of bounds\n");

}

int two_d_fetch (char *p, int size, int n, int m, int rows, int
columns) {
	 //printf("%d\n", p[rows * m + n]); //column major
	 //printf("%d\n", p[columns * n + m]);//Row major
	 return p[rows * m + n];
 }

void two_pd_store (char *p, int size, int n, int m, int rows, int
columns, int *value) {
	if (n < rows && n >= 0 && m < columns && m >= 0) {
		if (size <= sizeof(value)) {
			p[columns * n + m] = *value;     //Row major

		}
	}
```

```
      else
            printf("out of bounds\n");
}

int *two_pd_fetch (char *p, int size, int n, int m, int rows, int
columns, int *value) {
      int *arrayAdress;
      int val = p[rows *n + m];
      arrayAdress = &val;

      return arrayAdress;
}

int main () {
      int rows = 3;
      int columns = 4;
      int size = sizeof(int);
      int *ip;
      int value = 5;
      ip = &value;
      char *p = two_d_alloc(rows,columns,size);

      //two_pd_store(p,size,2,2,rows,columns,5);
      //two_d_store(p,size,0,2,rows,columns,3);
      //two_d_store(p,size,2,2,rows,columns,8);


      two_pd_store(p,size,1,1,rows,columns,ip);


      //printf("%d\n", two_pd_fetch(p,size,2,2,rows,columns));

      two_d_dealloc(p);
      return 0;

}
```

Task 2 (Memory dump)

For this exercise the previous char array implementation was used.
The output will be the numbers from 1 – 20 in hexadecimal form grouped in four words per line and the first of each row is the memory address of the first byte.
The if-statement serves to purposes it begins an new row when the iterator is dividable with 4 that way we get four words per row and after it begins a new row it prints the memory address of the first byte. For every iteration we print the hexadecimal notation of the value in the array.

For the last part there was some trouble translating to ASCII characters, and we didn't find a solution to how to check if the output was not a valid ASCII character.

```
#include <stdio.h>
#include <stdlib.h>
char * two_d_alloc (int n, int m, int size) {
	char *two_d_array = (char *)malloc(n * m * size);
	return two_d_array;
}
int main() {
	int rows = 3;
	int columns = 4;
	int size = sizeof(int);
	char *p = two_d_alloc(rows,columns,size);

	for (int i = 0; i < 20; i++) {
		p[i] = i;
		if ( i % 4 == 0 ) {
			printf("\n%p ", &p[i]);
		}
			printf("%08x ", p[i]);
		}
/*for (int j = 0; j < 20; j++) {
		printf("%c\n", (char)p[j]);
	}*/

	return 0;
}
```