Datateknik
Lab 3: Memory Organization

Alfred Elfving
Fahad Rejabo

Materials

• ThinLinc client on windows PC
• ThinLinc server running Ubuntu

Task 1: Array Storage

In this task we had to implement storage and fetching to an array in C code. We defined seperate functions to handle the array in a way that would seem to resemble a memory in hardware and how it operates.

The functions that we implemented in the code are the following:

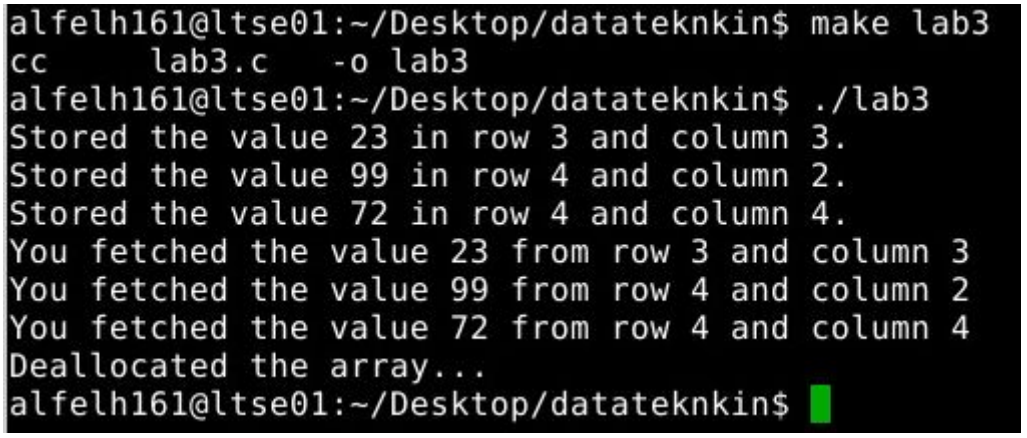
- *two_d_alloc*, to allocate memory for our storage.
- *two_d_dealloc*, to deallocate (free) memory that we allocated.
- *two_d_store*, to insert data into the storage.
- *two_d_fetch*, to retrieve data from our storage.

We have also implemented condition checking so the user cannot fetch or store outside of the allocated memory. For example, if a user has allocated a storage with width 10, then the user cannot store or fetch data wider than that width…
We have also defined our storage to use a row-major form, although we also have created a likewise program with column-major form, we will only focus on the row-major version in this report.

lab3.c - the code for the program

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

char* two_d_alloc(const int rows, const int columns, int element_size){
	char **arr = malloc(rows * columns * element_size);
	for (int i = 0; i < rows; i++)
		*arr = (char*)malloc(element_size);
	return *arr;
}


void two_d_dealloc(char* char_array){
	free(char_array);
	printf("Deallocated the array...\n");
}


void two_d_store(int value, char * arr_addr, int element_size, int row, int column, int amount_rows, int amount_columns){
	if (column < amount_columns && row < amount_rows){
		char* tmp_pointer = arr_addr;
		for(int i= 0;i<= row; i++){
			for (int j = 0;j<= column; j++){
				tmp_pointer++;
			}
		}
		*tmp_pointer = value;
		printf("Stored the value %d in row %d and column %d.\n", value, row, column);
	}else{
		printf("You cannot store something outside of the array!\n");
	}
}


int two_d_fetch(int row, int column, int amount_rows, int amount_columns, char * arr_addr, int size){
	if (row < amount_rows && column < amount_columns) {
		int fetched_value = 0;
		char* tmp_pointer = arr_addr;
		for(int i= 0;i<= row; i++){
			for (int j = 0;j<= column; j++){
				tmp_pointer++;
			}
		}
		fetched_value = *tmp_pointer;
		printf("You fetched the value %d from row %d and column %d\n", fetched_value, row, column);
	}else{
		printf("You cannot fetch something outside of the array!\n");
	}

	return 0;
}


int main(){
	//allocate 2D array
	char* arr = two_d_alloc(5,5,sizeof(char));
	// store values
	two_d_store(23, arr, sizeof(char), 3, 3, 5, 5);
	two_d_store(99, arr, sizeof(char), 4, 2, 5, 5);
	two_d_store(72, arr, sizeof(char), 4, 4, 5, 5);
	// fetch values
	two_d_fetch(3, 3, 5, 5, arr, sizeof(int));
	two_d_fetch(4, 2, 5, 5, arr, sizeof(int));
	two_d_fetch(4, 4, 5, 5, arr, sizeof(int));
	// deallocate array
	two_d_dealloc(arr);

	return 0;
}
```

The output of execution of the program



As we can see in the image that shows the output of the execution of the code, it executes properly without errors and behaves as we expected and planned. It simply creates a memory with a width and height of 5. We then simply store some values into the memory and fetch the same values. We end the program with deallocating the memory as well.

Task 2: Memory Dump

In this task we also implemented code in C that will do a memory dump of a character array. Task 2 is about to Implement a memory dump program which can read through a character array and prints out each word in hexadecimal notation.
In the code we go through each character in each string and also go through the bytes in the array (in the first loop), and the second loop is to go through each of the strings. In the code below we use “i” to go through each byte in our array, and “x” to go through each string, and the “size” is to go between cells, as each cell has “size” amount of bytes. Notice that we use “%02X” instead of “%X” because it print each character in 2 hex numbers, what i want to say is that we want to make sure we go 8 hexadecimal numbers for each string.

lab3.2.c file:

```
void memory_dump(char *array, int size, int rows, int col){

   // go through the byttes in the array
  for (int i = 0; i < rows*col*size; i=i+size){
     //four words per line
     if (i % (4*size) == 0)
     printf("\n %p -- ", &array[i]);
     //printf("Addres: %p,  hexa value: ", &array[i]);

   //go through each string
   for(int x=0; x < size; x++){
      //if (array[i+x] % 4 == 0)
```

```
      printf("%02X", array[x+i] & 0xff);
      //printf ("Addres: %p, hex value: %x\n", array[i+x],array[i+x]);
   }
      printf("--");
}
}
```

```
 0x7ffc9b0a0920 -- 00000000--00000000--00000000--00000000--
 0x7ffc9b0a0930 -- 01000000--00000000--6D074000--08000000--
 0x7ffc9b0a0940 -- 04000000--08000000--07000000--00000000--
 0x7ffc9b0a0950 -- 20090A9B--FC7F0000--00FE805E--179CA9BE--
 0x7ffc9b0a0960 -- 500A0A9B--FC7F0000--00000000--00000000--
 0x7ffc9b0a0970 -- 20074000--00000000--30B8F587--5E7F0000--
 0x7ffc9b0a0980 -- 00000000--00000000--580A0A9B--FC7F0000--
 0x7ffc9b0a0990 -- 00000000--01000000--59064000--00000000--fahreh161@ltse
01:~/Desktop/DBS-labar$
```

The output of the code