Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

Computer Science Laboration 2

Task 1

In task 1 we were instructed to create the functions described below and modify the functions to work for a general case of input types (e.g. double and node). varType in image 1 can be modified to the input type and the functions will work accordingly.

Two_de_alloc

This function allocates space in memory of a simulated two dimensional NxM matrix where each “slot” in the matrix is size bytes. The function then returns an array pointer to the allocated memory.

Two_d_dealloc

This function free’s the allocated memory where the array is pointing.

Two_d_store

This function stores argument varType *arg in a specified position in the 2D array using row-major form, where varType *arg is an address pointer. We use matrix arithmetics to decide what position to store the argument.

Two_d_store_col

This function works in the same way as two_d_store but stores using column-major form.

The functions above can be seen in image 1.

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

typedef double varType;

char *two_d_alloc(int n, int m, int size) {
        char *array = (char *)malloc(n*m*size);

        return array;
}

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

void two_d_store(varType *arg, char *arr, int size, int row, int col, int n,
                int m) {
        *(varType *)(arr+(row*m+col)*size) = *arg;
}

void two_d_store_col(varType *arg, char *arr, int size, int row, int col, int n,
                int m) {
        *(varType *)(arr+(row+n*col)*size) = *arg;
}
```

Image 1

Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

Two_d_fetch

This function fetches an address pointer stored in the requested position using row-major form. We access the position to fetch from in the same way as we do in store, with the help of matrix arithmetics.

Two_d_fetch_col

This function works as tow_d_fetch but uses column-major form

The functions mentioned above can be seen in image 2

```
varType *two_d_fetch(int row, int col, int n, int m, char *arr, int size) {
        varType *value = (varType *)(arr+(row*m+col)*size);

        return value;
}

varType *two_d_fetch_col(int row, int col, int n, int m, char *arr, int size) {
        varType *value = (varType *)(arr+(row+n*col)*size);

        return value;
}
```

Image 2

Main

In the main we test our functions for a 2D array. We store then fetch a number of doubles. The rest of this section will show how we implement our tests.

In image 3 you can see how we implement and store the char pointer for an example matrix that is 2 x 3 in size. We also set the argument to be -12.6 (the snippet is and old screenshot, see image 7 for the output when arg is -12.6.

```
int main () {
        int n = 2;
        int m = 3;
        int size = sizeof(varType);

        //Allocates memory for a n * m array
        char *array = two_d_alloc(n,m,size);
        char *array_col = two_d_alloc(n,m,size);

        int i,j;
        varType arg = 12.6;
```

Image 3

Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

With the for loop in image 4 we store the argument in position [i,j]. We run two_d_store, two_d_store_col at the same time to save runtime.

```
int i,j;
varType arg = 12.6;

for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
                two_d_store(&arg, array, size, i, j, n, m);
                two_d_store_col(&arg, array_col, size, i, j, n, m);
                arg = arg + 1.5;
        }
}

printf("Row-major form\n");
for(i = 0; i < n; i++){
        for(j = 0; j < m; j++) {
        printf("Value at pos [%d %d] is: %g\n",
                        i + 1,j + 1,
                        *(two_d_fetch(i,j,n,m, array, size)));
```

Image 4

In image 5 our code for implementing fetch is shown. We use the both fetch functions in a printf function so we can see the output. To see the output of this code look at image 7

```
printf("Row-major form\n");
for(i = 0; i < n; i++){
        for(j = 0; j < m; j++) {
        printf("Value at pos [%d %d] is: %g\n",
                        i + 1,j + 1,
                        *(two_d_fetch(i,j,n,m, array, size)));
        }
}

printf("\nColumn-major form\n");
for(i = 0; i < n; i++){
        for(j = 0; j < m; j++) {
        printf("Value at pos [%d %d] is: %g\n",
                        i + 1, j + 1,
                        *(two_d_fetch(i,j,n,m, array_col, size)));
        }
}

// Here we deallocate the array from the memory
```

Image 5

In image 6 we use our function two_d_dealloc to deallocate the two arrays.

```
// Here we deallocate the array from the memory
two_d_dealloc(array);
two_d_dealloc(array_col);

return 0;
}
```

Image 6

Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

We compile the program with this line “gcc main.c -o main”, then we run the program with “./main”. Then you can see the output of the program. We tested the boundary conditions by using a negative argument and it still works as we expected it to.

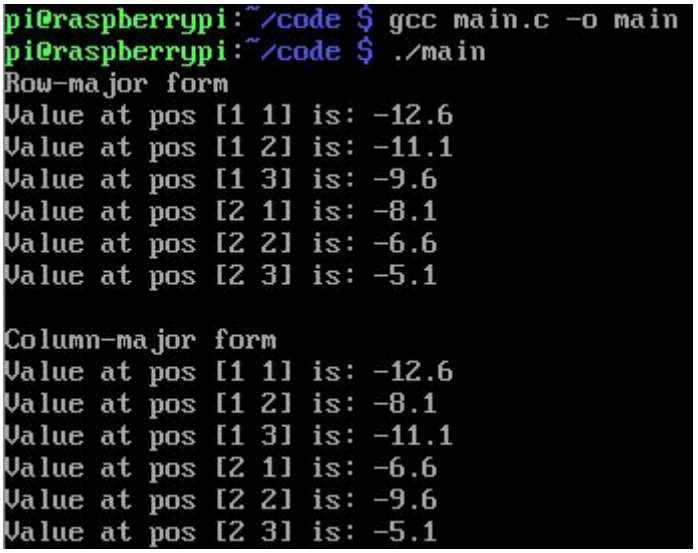

Image 7