Datorteknik, HT18, DT509G
Lab 3: Memory Organization

Kinfai Chin & Karl Eriksson

2018-10-12

Task 1: Array Storage (15 points)

The task was to write various functions and structures in order to store and fetch from a 2D byte array. The task also stated that a 1D char* array was to be used to emulate an 2D array in C.

Step 1
The function two_d_alloc takes in number of rows, columns and the element size in order to allocate enough memory using malloc for each slot in the array. (Codeline 4).

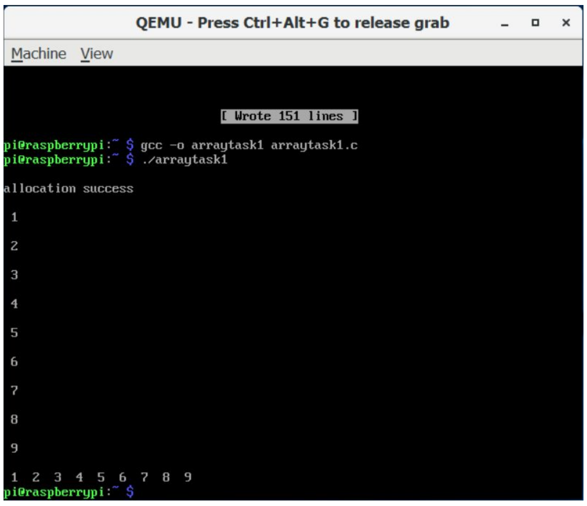
Step 2
The function two_d_dealloc deallocates the the array by using free(). (Codeline 24)

Step 3
Check main() at codeline 139.

Step 4
The function two_d_store takes in the array, the row and column which the new value is to be stored, the matrix size and the size of each arrayslot. The position of the value in the matrix will be converted via ColumnMax*Row+Column to get the right position in the 1D array. (Codeline 29)

Step 5
The function two_d_fetch takes in the array, column and row number, it also takes in the size of each element and the amount of columns and rows the matrix has in total in order to be able to calculate position. It then takes out the value that is stored in that position and returns it.

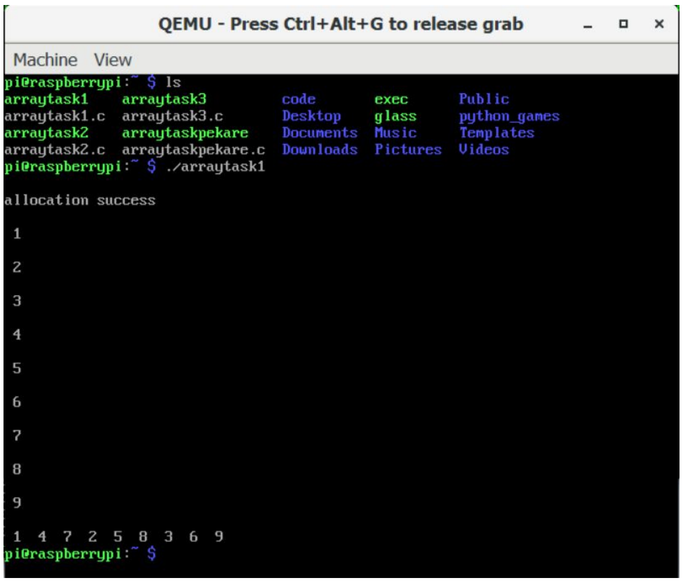
Step 6
The init_loop function runs the store and fetch on a 3x3 matrix. (Codeline 92)



```
                    [ Wrote 151 lines ]

pi@raspberrypi:~ $ gcc -o arraytask1 arraytask1.c
pi@raspberrypi:~ $ ./arraytask1

allocation success

 1

 2

 3

 4

 5

 6

 7

 8

 9

 1  2  3  4  5  6  7  8  9
pi@raspberrypi:~ $
```

Step 7
In order to counteract problems the store and fetch functions has if-statements that checks if the user is trying to store elements outside of the matrix boundaries.

Step 8
Two more functions were written. two_d_store_col (Codeline 66) and two_d_fetch_col (Codeline 69). Another initialization loop (init_loop_col Codeline 115) was written to check that everything works as intended. As seen on the bottom row (elements printed in order of the array) we can see that the printout follows the column major format.



```
pi@raspberrypi:~ $ ls
arraytask1    arraytask3         code      exec      Public
arraytask1.c  arraytask3.c       Desktop   glass     python_games
arraytask2    arraytaskpekare    Documents Music     Templates
arraytask2.c  arraytaskpekare.c  Downloads Pictures  Videos
pi@raspberrypi:~ $ ./arraytask1

allocation success

 1

 2

 3

 4

 5

 6

 7

 8

 9

 1  4  7  2  5  8  3  6  9
pi@raspberrypi:~ $
```

Step 9
The function arguments and return values were adjusted to accommodate memory address instead of taking in and returning the actual value. The functions structure is the same except for that the memory address is used instead of returning actual values.

```
28 . void two_d_store(char *p, int n, int m, int *value, int N, int M, int size){ .
 .

 .
45 . char* two_d_fetch(char *p,int n,int m, int N, int M, int size){
  .
  .
95 .     two_d_store(array, 0, 0,((int*)&a), n, m,size);
96 .     two_d_store(array, 1, 0,((int*)&b), n, m,size);
```

Step 10

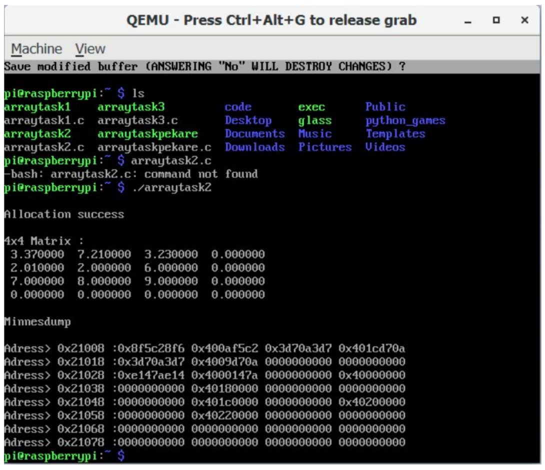
To make sure that other data types also work with inserting and fetching values from the array each step takes into consideration how big the datatype is. For example if you insert data type int which consists of 4 bytes it jumps 4 bytes instead of just jumping one byte like before.

```c
void two_d_store(char *p, int n, int m, int *value, int N, int M, int size){
    if(n >= N){
            printf("\ndimensions doesnt match\n");
            return;
    }
    if(m >= M){
            printf("\ndimensions doesnt match\n");
            return;
    }
    //printf("Value: %d\n", *value);
    int i;
    for(i = 0;i < size; i++){
            p[(N * m + n)*size + i] = *((char*)value+i);
    }
    return;
}

char* two_d_fetch(char *p,int n,int m, int N, int M, int size){
    if(n >= N){
            printf("\nElement does not exist\n");
            return;
    }
    if(m >= M){
            printf("\nElement does not exist\n");
            return;
    }
    int position = (N * m + n)*size;
    char *x = p+position;
    //printf("%d ", (int)*x);
    return x;
}
```

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

char* two_d_alloc(int N, int M, int size){

    //allocates an char array with size n*m
    //is acutally an 1D array using buffer

    char *c_array;

    //calls malloc to allocate appropriate number of bytes for the array
    c_array = (char *) malloc(size*N*M);

    if(c_array != NULL){
            printf("\nallocation success\n");
    }
    else{
            printf("\nallocation fail\n");
    }
    return c_array;

}

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

void two_d_store(char *p, int n, int m, int value, int N, int M, int size){
    if(n >= N){
            printf("\ndimensions doesnt match\n");
            return;
    }
    if(m >= M){
            printf("\ndimensions doesnt match\n");
            return;
    }
    p[N * m + n] = value;
    return;
}

int two_d_fetch(char *p,int n,int m, int N, int M, int size){
    if(n >= N){
            printf("\nelement does not exist\n");
            return;
    }
    if(m >= M){
            printf("\nelement does not exist\n");
            return;
    }
    int x = p[N * (m) + n];
    printf("\n %d \n",x);
    return p[n*m];
}

void two_d_store_col(char *p, int n, int m, int value, int N, int M, int size){
    if(n >= N){
            printf("\ndimensions doesnt match\n");
            return;
    }
    if(m >= M){
            printf("\ndimensions doesnt match\n");
            return;
    }
    p[M * n + m] = value;
    return;
}

int two_d_fetch_col(char *p,int n,int m, int N, int M, int size){
    if(n >= N){
            printf("\nelement does not exist\n");
            return;
    }
    if(m >= M){
            printf("\nelement does not exist\n");
            return;
    }
    int x = p[M * n + m];
    printf("\n %d \n",x);
    return p[n*m];
}
int loop(char* p){
    int i = 0;
    printf("\n");
    while ( i < 9){
            printf(" %d ",p[i]);
            i++;
    }
    printf("\n");
}

int init_loop(char *array, int n , int m){
    two_d_store(array, 0, 0, 1, n, m,sizeof(int));
    two_d_store(array, 1, 0, 2, n, m,sizeof(int));
    two_d_store(array, 2, 0, 3, n, m,sizeof(int));
    two_d_store(array, 0, 1, 4, n, m,sizeof(int));
    two_d_store(array, 1, 1, 5, n, m,sizeof(int));
    two_d_store(array, 2, 1, 6, n, m,sizeof(int));
    two_d_store(array, 0, 2, 7, n, m,sizeof(int));
    two_d_store(array, 1, 2, 8, n, m,sizeof(int));
    two_d_store(array, 2, 2, 9, n, m,sizeof(int));

    two_d_fetch(array, 0, 0, n, m,sizeof(int));
    two_d_fetch(array, 1, 0, n, m,sizeof(int));
    two_d_fetch(array, 2, 0, n, m,sizeof(int));
    two_d_fetch(array, 0, 1, n, m,sizeof(int));
    two_d_fetch(array, 1, 1, n, m,sizeof(int));
    two_d_fetch(array, 2, 1, n, m,sizeof(int));
    two_d_fetch(array, 0, 2, n, m,sizeof(int));
    two_d_fetch(array, 1, 2, n, m,sizeof(int));
    two_d_fetch(array, 2, 2, n, m,sizeof(int));
    return;
}

int init_loop_col(char *array, int n , int m){
    two_d_store_col(array, 0, 0, 1, n, m,sizeof(int));
    two_d_store_col(array, 1, 0, 2, n, m,sizeof(int));
    two_d_store_col(array, 2, 0, 3, n, m,sizeof(int));
    two_d_store_col(array, 0, 1, 4, n, m,sizeof(int));
    two_d_store_col(array, 1, 1, 5, n, m,sizeof(int));
    two_d_store_col(array, 2, 1, 6, n, m,sizeof(int));
    two_d_store_col(array, 0, 2, 7, n, m,sizeof(int));
    two_d_store_col(array, 1, 2, 8, n, m,sizeof(int));
    two_d_store_col(array, 2, 2, 9, n, m,sizeof(int));

    two_d_fetch_col(array, 0, 0, n, m,sizeof(int));
    two_d_fetch_col(array, 1, 0, n, m,sizeof(int));
    two_d_fetch_col(array, 2, 0, n, m,sizeof(int));
    two_d_fetch_col(array, 0, 1, n, m,sizeof(int));
    two_d_fetch_col(array, 1, 1, n, m,sizeof(int));
    two_d_fetch_col(array, 2, 1, n, m,sizeof(int));
    two_d_fetch_col(array, 0, 2, n, m,sizeof(int));
    two_d_fetch_col(array, 1, 2, n, m,sizeof(int));
    two_d_fetch_col(array, 2, 2, n, m,sizeof(int));
    return;
}


int main(){
    int n = 3;
    int m = 3;
    //n  = kolumn
    //m  = rad
    //index start n = 0, m = 0
    char *array = two_d_alloc(n,m,sizeof(char));
    //init_loop(array,n,m);
    init_loop_col(array, n , m);
    loop(array);
    two_d_dealloc(array);
    return 0;
}

```

Task 2: Memory Dump (5 points)

The assignment was to read through the array and print out the address of the first word then print the value of each word with four words per line. The words should also be displayed in hexadecimal notation. (Codeline 85)



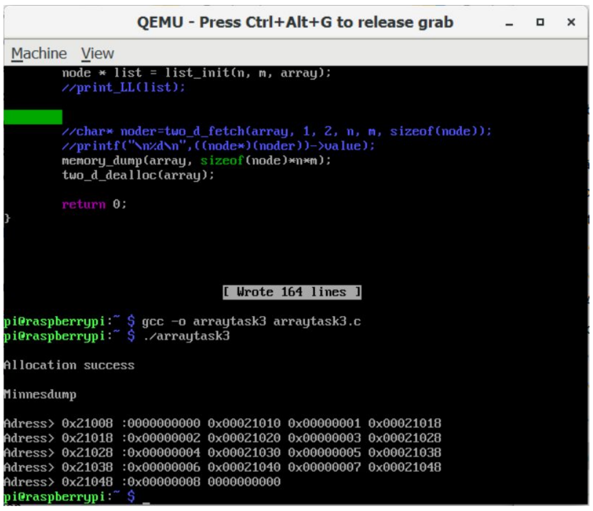
```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?

pi@raspberrypi:~ $ ls
arraytask1    arraytask3         code       exec      Public
arraytask1.c  arraytask3.c       Desktop    glass     python_games
arraytask2    arraytaskpekare    Documents  Music     Templates
arraytask2.c  arraytaskpekare.c  Downloads  Pictures  Videos
pi@raspberrypi:~ $ arraytask2.c
-bash: arraytask2.c: command not found
pi@raspberrypi:~ $ ./arraytask2

Allocation success

4x4 Matrix :
 3.370000  7.210000  3.230000  0.000000
 2.010000  2.000000  6.000000  0.000000
 7.000000  8.000000  9.000000  0.000000
 0.000000  0.000000  0.000000  0.000000

Minnesdump

Adress> 0x21008 :0x8f5c28f6 0x400af5c2 0x3d70a3d7 0x401cd70a
Adress> 0x21018 :0x3d70a3d7 0x4009d70a 0000000000 0000000000
Adress> 0x21028 :0xe147ae14 0x4000147a 0000000000 0x40000000
Adress> 0x21038 :0000000000 0x40180000 0000000000 0000000000
Adress> 0x21048 :0000000000 0x401c0000 0000000000 0x40200000
Adress> 0x21058 :0000000000 0x40220000 0000000000 0000000000
Adress> 0x21068 :0000000000 0000000000 0000000000 0000000000
Adress> 0x21078 :0000000000 0000000000 0000000000 0000000000
pi@raspberrypi:~ $
```

```
1  . #include <stdlib.h>
2  . #include <stdio.h>
3  .
4  . char* two_d_alloc(int N, int M, int size){
5  .
6  .     //allocates an char array with size n*m
7  .     //is acutally an 1D array using buffer
8  .
9  .     char *c_array;
10 .
11 .     //calls malloc to allocate appropriate number of bytes for the array
12 .     c_array = (char *) malloc(size*N*M);
13 .
14 .     if(c_array != NULL){
15 .             printf("\nAllocation success\n");
16 .     }
17 .     else{
18 .             printf("\nAllocation fail\n");
19 .     }
20 .     return c_array;
21 .
22 . }
23 .
24 . void two_d_dealloc(char *p){
25 .     free(p);
26 .     return;
27 . }
28 . void two_d_store(char *p, int n, int m, int *value, int N, int M, int size){
29 .     if(n >= N){
30 .             printf("\ndimensions doesnt match\n");
31 .             return;
32 .     }
33 .     if(m >= M){
34 .             printf("\ndimensions doesnt match\n");
35 .             return;
36 .     }
37 .     //printf("Value: %d\n", *value);
38 .     int i;
39 .     for(i = 0;i < size; i++){
40 .             p[(N * m + n)*size + i] = *((char*)value+i);
41 .     }
42 .     return;
43 . }
44 .
```

```c
char* two_d_fetch(char *p,int n,int m, int N, int M, int size){
    if(n >= N){
            printf("\nElement does not exist\n");
            return;
    }
    if(m >= M){
            printf("\nElement does not exist\n");
            return;
    }
    int position = (N * m + n)*size;
    char *x = p+position;
    //printf("%d ", (int)*x);
    return x;
}
void print_matrix(char *p,int n, int m){
    printf("\n%dx%d Matrix : \n", n , m);
    int i = 0;
    int j = 0;
    while (j < n){
            while(i < m){
                    char *x = two_d_fetch(p,i,j,n,m,sizeof(double));
                    printf(" %f ", *(double*)x);
                    i++;
                    }
            printf("\n");
            j++;
            i = 0;
    }
}

int loop(char* p){
    int i = 0;
    printf("Array printed in order of elements : \n");
    while ( i < 9*4){
            printf(" %d ",p[i]);
            i++;
    }
    printf("\n");
}

void memory_dump(char* array, int numbytes){
    printf("\nMinnesdump\n");
    int k;
    int siffra = 0;
    for(k = 0; k < numbytes/4;k++){ //divides the total amount of bytes by 4 because were casting to int
            siffra = *(((int*)array) + k);
            int *adressen = (((int*) array) + k);
            if(k % 4 == 0){
                    printf("\nAdress> %p :",adressen);
            }
            printf("%#010x ", siffra);
    }
    printf("\n");
}

int init_loop(char *array, int n , int m, int size){
    double a = 3.37;
    double b = 7.21;
    double c = 3.23;
    double d = 2.01;
    double e = 2;
    double f = 6;
    double g = 7;
    double h = 8;
    double i = 9;
    two_d_store(array, 0, 0,((int*)&a), n, m,size);
    two_d_store(array, 1, 0,((int*)&b), n, m,size);
    two_d_store(array, 2, 0,((int*)&c), n, m,size);
    two_d_store(array, 0, 1,((int*)&d), n, m,size);
    two_d_store(array, 1, 1,((int*)&e), n, m,size);
    two_d_store(array, 2, 1,((int*)&f), n, m,size);
    two_d_store(array, 0, 2,((int*)&g), n, m,size);
    two_d_store(array, 1, 2,((int*)&h), n, m,size);
    two_d_store(array, 2, 2,((int*)&i), n, m,size);
    two_d_fetch(array, 0, 0, n, m,size);
    two_d_fetch(array, 1, 0, n, m,size);
    two_d_fetch(array, 2, 0, n, m,size);
    two_d_fetch(array, 0, 1, n, m,size);
    two_d_fetch(array, 1, 1, n, m,size);
    two_d_fetch(array, 2, 1, n, m,size);
    two_d_fetch(array, 0, 2, n, m,size);
    two_d_fetch(array, 1, 2, n, m,size);
    two_d_fetch(array, 2, 2, n, m,size);
    return;
}

int main(){
    //n  = kolumn
    //m  = rad
    //index start n = 0, m = 0
    int n = 4;
    int m = 4;
    int size = sizeof(double);
    char *array = two_d_alloc(n,m,size);
    //two_d_dealloc(array);

    init_loop(array,n,m,size);
    print_matrix(array,n,m);
    memory_dump(array, n*m*size);
    //loop(array);
    return 0;
}
```

Task 3: Linked Lists (5 points)

The following task was to implement a struct which holds an element value and a pointer to the next node in the list. The nodes of the struct were then placed in the array in corresponding order. The memory dump shows the address and value in hexadecimal form. As shown below the value and address of every element in the linked list is printed. The address of each element is incremented and judging by what our memory dump has printed out we can see that each node points at the next one and that the last one points at NULL (00000000000000). (Codeline 24)



```
        node * list = list_init(n, m, array);
        //print_LL(list);


        //char* noder=two_d_fetch(array, 1, 2, n, m, sizeof(node));
        //printf("\n%d\n",((node*)(noder))->value);
        memory_dump(array, sizeof(node)*n*m);
        two_d_dealloc(array);

        return 0;
}




                              [ Wrote 164 lines ]

pi@raspberrypi:~ $ gcc -o arraytask3 arraytask3.c
pi@raspberrypi:~ $ ./arraytask3

Allocation success

Minnesdump

Adress> 0x21008 :0000000000 0x00021010 0x00000001 0x00021018
Adress> 0x21018 :0x00000002 0x00021020 0x00000003 0x00021028
Adress> 0x21028 :0x00000004 0x00021030 0x00000005 0x00021038
Adress> 0x21038 :0x00000006 0x00021040 0x00000007 0x00021048
Adress> 0x21048 :0x00000008 0000000000
pi@raspberrypi:~ $ _
```

```c
#include <stdlib.h>
#include <stdio.h>
typedef struct element{
    int value;
    struct element * next;
}node;
void two_d_store(char *p, int n, int m, int *value, int N, int M, int size){
    if(n >= N){
            printf("\ndimensions doesnt match\n");
            return;
    }
    if(m >= M){
            printf("\ndimensions doesnt match\n");
            return;c
    }
    //printf("Value: %d\n", *value);
    int i;
    for(i = 0;i < size; i++){
            p[(N * m + n)*size + i] = *((char*)value+i);
    }
    return;
}

node * list_init(int n, int m, char *array){
    node * head;
    int elements =  n*m;
    head = (node *)(array);
    head->value = 0;
    head->next = NULL;
    two_d_store(array, 0, 0, (int*)head, n, m, sizeof(node));
    int i;
    add_after(head, 1, 1, 0, array);
    add_after(head, 2, 2, 0, array);
    add_after(head, 3, 0, 1, array);
    add_after(head, 4, 1, 1, array);
    add_after(head, 5, 2, 1, array);
    add_after(head, 6, 0, 2, array);
    add_after(head, 7, 1, 2, array);
    add_after(head, 8, 2, 2, array);
    return head;
}

int add_after(node *list, int data, int n, int m, char *array){
    node * current = list;

    while(current->next != NULL){
            current = current->next;
    }
    current->next = (node *)(array+(3 * m + n) * sizeof(node));
    //two_d_store(array, n, m, (int*)(current->next), 3, 3, sizeof(node));
    current->next->value = data;
    current->next->next = NULL;
    return 0;
}



char* two_d_alloc(int N, int M, int size){

    //allocates an char array with size n*m
    //is acutally an 1D array using buffer

    char *c_array;

    //calls malloc to allocate appropriate number of bytes for the array
    c_array = (char *) malloc(size*N*M);

    if(c_array != NULL){
            printf("\nAllocation success\n");
    }
    else{
            printf("\nAllocation fail\n");
    }
    return c_array;

}

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


char* two_d_fetch(char *p,int n,int m, int N, int M, int size){
    if(n >= N){
            printf("\nElement does not exist\n");
            return;
    }
    if(m >= M){
            printf("\nElement does not exist\n");
            return;
    }
    int position = (N * m + n)*size;
    char *x = p+position;
    //printf("%d ", (int)*x);
    return x;
}
void print_matrix(char *p,int n, int m){
    printf("\n%dx%d Matrix : \n", n , m);
    int i = 0;
    int j = 0;
    while (j < n){
            while(i < m){
                    char *x = two_d_fetch(p,i,j,n,m,sizeof(double));
                    printf(" %f ", *(double*)x);
                    i++;
                    }
            printf("\n");
            j++;
            i = 0;
    }
}

int loop(char* p){
    int i = 0;
    printf("Array printed in order of elements : \n");
    while ( i < 9*4){
            printf(" %d ",p[i]);
            i++;
    }
    printf("\n");
}

void memory_dump(char * array, int numbytes){
    printf("\nMinnesdump\n");
    int k;
    int siffra = 0;
    for(k = 0; k < numbytes/4;k++){
            siffra = *(((int*)array )+ k);
            int *adressen = (((int*) array) + k);
            if(k % 4 == 0){
                    printf("\nAdress> %p :",adressen);
            }
            printf("%#010x ", siffra);
    }
    printf("\n");
}

void print_LL(node * head){
    while(head != NULL){
            printf("\n%d\n",head->value);
            head = head->next;
    }
}

int main(){
    //n  = kolumn
    //m  = rad
    //index start n = 0, m = 0
    int n = 3;
    int m = 3;

    char* array = two_d_alloc(n,m,sizeof(node));
    node * list = list_init(n, m, array);
    //print_LL(list);


    //char* noder=two_d_fetch(array, 1, 2, n, m, sizeof(node));
    //printf("\n%d\n",((node*)(noder))->value);
    memory_dump(array, sizeof(node)*n*m);
    two_d_dealloc(array);

    return 0;
}

```