Task 1:

```c
char* two_d_alloc(int rows,int columns,int size)
{
    printf("creating...\n");
    char *str;
    str = (char *) malloc((rows*columns*size));

    str[rows*columns*size] = '\0';

    return str;
}
```

```c
void two_d_dealloc(char *trash)
{
    printf("in trash-can.\n");
    free(trash);
}
```

The alloc function and the dealloc function

```c
int main(){
    int rows = 3;
    int columns = 3;
    int size = 1;

    char* head = two_d_alloc(rows,columns,size);
    head = two_d_store(head,rows,columns,3,1,4);


    printMatrix(rows,columns,head);
```

the main function to test all exercises. the printMatrix function prints out the given matrix.

```c
char* two_d_store(char *array, int rows, int columns, int r, int c, int input)
{   //row-major format
    if(r<=rows && c<=columns){
        array[rows*(r-1) + c -1] = input + '0';
    }
    return array;
}

int two_d_fetch(char *array, int rows, int columns, int r, int c)
{
    return (array[rows*(r-1) + c -1] - '0');
}
```

the store and fetch function.

```
    char* head = two_d_alloc(rows,columns,size);
    head = two_d_store(head,rows,columns,3,1,4);


    printMatrix(rows,columns,head);

    int fetch = two_d_fetch(head,rows,columns,3,1);
    printf("%d\n",fetch);
```

this is a part from the main function. It allocates space for a 3x3 matrix, store the element ”4” in row 3 and column 1, then fetches it and prints it. When I try to input an element into a space which does not exist in the given matrix, my program doesnt store anything because of the if-statement in the store function.

```
char * two_d_storeColumn(char *array, int rows, int columns, int r, int c, int input)
{//column-major format
    array[columns*(c-1) + r -1] = input + '0';
    return array;
}



int two_d_fetchColumn(char *array, int rows, int columns, int r, int c)
{
    return (array[rows*(r-1) + c -1] - '0');
}
```

two_d_store and two_d_fetch in column major format.

```
char* two_d_storePointer(char *array, int rows, int columns, int r, int c, int *adress)
{
    //printf("before: %p\n",adress);
    //printf("input: %c\n",*input);
    array[rows*(r-1) + c -1] = *adress + '0';
    //memcpy(&array[rows*(r-1) + c -1],&input,1);
    //printf("yet: %p\n",&array[rows*(r-1) + c -1]);
    //array[rows*(r-1) + c -1] = *adress;


    return array;
}

char* two_d_fetchPointer(char *array, int rows, int columns, int r, int c)
{
    return (&array[rows*(r-1) + c -1]);
    //int (*adress) = array[rows*(r-1) + c -1];

}
```

this is a fetch and store that operates on integer pointers instead of by value. they are implemented in row-major format because it’s easier to understand for me.

I test all exercises in task one with this main function:

```
int main(){
    int rows = 3;
    int columns = 3;
    int size = 1;

    char* head = two_d_alloc(rows,columns,size);
    head = two_d_store(head,rows,columns,3,1,4);


    printMatrix(rows,columns,head);

    int fetch = two_d_fetch(head,rows,columns,3,1);
    printf("%d\n",fetch);

    two_d_dealloc(head);




    rows = 2;
    columns = 2;
    char* twoByTwo = two_d_alloc(rows,columns,size);
    twoByTwo = two_d_storeColumn(twoByTwo,rows,columns,2,1,6);


    int a = 30;
    int *b;
    b = &a;


    two_d_storePointer(twoByTwo,rows,columns,2,1,b);

    char *add = two_d_fetchPointer(twoByTwo,rows,columns,2,1);
    int fetch2 = *add -'0';
    printf("fetch: %d\n",fetch2);

    printMatrix(rows,columns,twoByTwo);


    return 0;
}
```

and the output is:

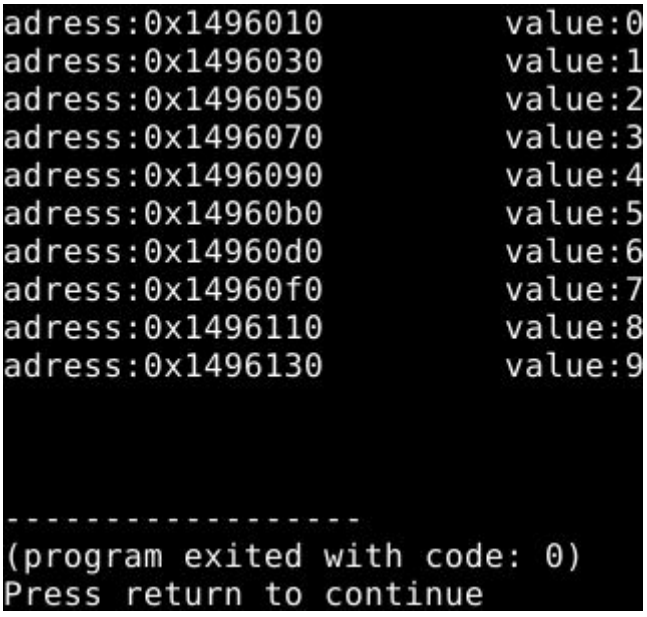
```
creating...

1:       ,2:      ,3:      ,
4:       ,5:      ,6:      ,
7:4      ,8:      ,9:      ,
4
in trash-can.
creating...
fetch: 30

1:       ,2:6     ,
3:30     ,4:      ,


------------------
(program exited with code: 0)
Press return to continue
```

Task 3:

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

struct Node
{
    // Any data type can be stored in this node
    int  data;

    struct Node *next;
};

void printList(struct Node *head){
    struct Node *start = head;

    while(start != NULL){

        printf("adress:%p\t value:%d\n",start,start->data);
        start = start->next;
    }
    printf("\n");
}


int main(){

    int size = 10;
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 0;

    struct Node* temp = head;

    for(int i=1; i<size ;i++){

        struct Node* element = (struct Node*)malloc(sizeof(struct Node));
        temp->next = element;
        temp->next->data = i;
        temp = temp->next;
    }

    printList(head);

return 0;
}
```

this is my entire code for task3.

the output of the program is:



we can see that the adress is incremented by 0x0000020 with every node.
this is the viritual memory address, so thats why we don’t see any special jump after 4 bytes (32 bits).