Matam – 234122
Homework #2 – dry part

1. Assert exercise:

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <malloc.h>
#define N 10
int f(char *);
int main(int argc, char **argv)
{
 /* 1 */
assert(argc==2); // wrong use of assert, it cannot be always equal to 2 because it is depending on the number of
                 //parameters inserted by the user at running. The success of assert couldn't relay on the user's input.

 char *s = malloc(N);
 /* 2 */
 assert(s!=NULL);  //wrong use of assert, we need to check the success of "malloc" function, not to assume it
                   //succeeded

scanf("%s", s);
 /* 3 */
assert(strlen(s)<N); //wrong use of assert, we get an input from the user, we can't assume nothing about its length.
                     // the "scanf" function can copy to "s" string even more then its allocated size (this is a famous bug)

 /* 4 */
assert(!*(s+strlen(s))); //good use of assert, it check if the "scanf" function indeed put NULL(\0) at the
                         //end of the input str.

 /* 5 */
assert(atol(s)); //wrong use of assert, if the "s" string doesn't include number it will return 0 and the assert will
                 //collapse, which means that the success of the assert depends on the user.
                 //"atoll" function gets string, and returns the integer value of the string(if the string is number)
                 //or zero(if the string doesn't have only numbers).

 printf("%ld\n", 100000000/atol(s));
 free(s);
 return 0;
}

int f(char *s)
{
 /* 6 */
 assert(s!=NULL); //there are two different cases(depends):
                  //- if we assume that the programmer that use this function knows that forbidden to send null
                      //pointer, this a good use of assert.
                  //- if we don't assume that, this assert could collapse by a bed parameter that the programmer using
                      //this function had sent to. This way that is a wrong use of assert.

 return !*s;
}
```

2. text_flat Function exercise:

```
#define NUM_OF_WORDS 4

// ===================== flat_text function ==========================
char *flat_text(char **words, int n){
    char* str = NULL;
    char* str_tmp = NULL;
    int len_of_str=1;
    int privious_str_len=0;
    for (int i=0; i<n; i++) {
        len_of_str += strlen(*(words+i));
        str = malloc(len_of_str*sizeof(char));
        if (str == NULL) {
            free(str_tmp);
            return str;
        }
        else if (i == 0) {
            strcpy(str, *words);
            str_tmp = str;
            privious_str_len = len_of_str;
        }
        else if (i != 0) {
            strcpy(str, str_tmp);
            strcpy((str+privious_str_len-1) ,*(words+i));
            free(str_tmp);
            str_tmp = str;
            privious_str_len = len_of_str;
        }
    }
    return str_tmp;
}

// ===================== main function ==========================
int main()
{
    char *words[NUM_OF_WORDS] = {"Hello" , "To", "234122" , "Matam"};
    char *p = flat_text(words, NUM_OF_WORDS);
    if(p != NULL) {
        printf("\n%s\n", p);
        free(p);
    }
    return 0;
}
```

- Output value of running flat_text function :