C

Algae Malloc

```
#include<stdio.h>  → malloc
#include<stdlib.h> → malloc
#include<string.h> → strcpy, strlen

void changing(char *algae)
{
    char * supporting_tab = malloc(1000 * sizeof(char));   // memory reservation for mem | table
                                                           // 1000 * 1B = 1kB (1000/1024 kiB)
    int i, j=0;   → iterators
    for(i=0; i<strlen(algae); i++)   → changing as for as the size of an array
    {
        if(algae[i]=='A')
        {
            supporting_tab[j]='A';
            supporting_tab[j+1]='B';
            j+=2;   → increasing sup_tab iterator
        }
        else if(algae[i]=='B')
        {
            supporting_tab[j]='A';
            j++;
        }
    }
    supporting_tab[j]='\0';   → first element is empty (security)
    strcpy(algae, supporting_tab);   → copying sup_tab content to algae tab
    free(supporting_tab);   → freeing up unnecessary memory
}
int main()
{
    char algae[1000]="A";   → first element of a 1000-element array is "A" (string)
    int i;
    for(i=0; i<=10; i++)
    {
        printf("%d. %s\n", i, algae);   → for example 2. ABA
        changing(algae);
    }
    return 0;
}
```

A → AB → ABA → ABABA → ...

A note for myself (polish language)

Algae pointers

#include <stdio.h>

#define SIZE 100 stały rozmiar tablicy znaków

funkcja „zamawiająca" pamięć z tablicy algae na inne znaki w

void dynamics (char *); tablicy pomocniczej, przyjmuje wskaźnik

int main()

{

char algae[SIZE]; deklaracja głównej tablicy algae

dynamics(algae); nazwa tablicy jest wskaźnikiem

return 0;

}

void dynamics (char *algae-tab)

{

char supporting-tab[200] = {'0'}; ; tablica pomocnicza o rozmiarze 200 dwoma elementami '0' i '0'

*algae-tab = 'A' pierwszy element = 'A'

printf("i.d. i.c \n", new-nr, *algae-tab); wypisane 0. elementu

new-nr++; teraz new-nr zaczyna się od 1.

od 1 do 10+1 powtórzeń

for(new-nr=1; new-nr < number-of-operations+1; new-nr++)

{

while(*algae + alg-iterator) != '0' dopóki nie osiągniemy pustego elementu

od dołu

```
for (new_row=1; new_row < number_of_operations +1; new_row ++)
{
  while (*(alpae_tab + alg_iterator) != '\0')
  {
    if (*(alpae_tab + alg_it.) == 'A')
    {
      sup_tab[sup_it] = 'A';
      sup_tab[sup_it +1] = 'B';
      sup_it += 2;
    }
    else if (*(alg_tab + alg_it) == 'B')
    {
      sup_tab[sup_it] = 'A';
      sup_it ++;
    }
    alg_it ++;
  }
  for (asig_it = 0; asp_it < SIZE; asp_it ++)
  {
    *(alg_tab + alg_it) = sup_tab[sup_it];
  }
    printf("%d", new_row);
  for (asig_it = 0; *(alg_tab + asig_it) != '\0'; asig_it ++)
  {
    printf("%c", *(alg_tab + asig_it));
  }
printf("\n");
```

dopóki nie napotkamy pustego miejsca

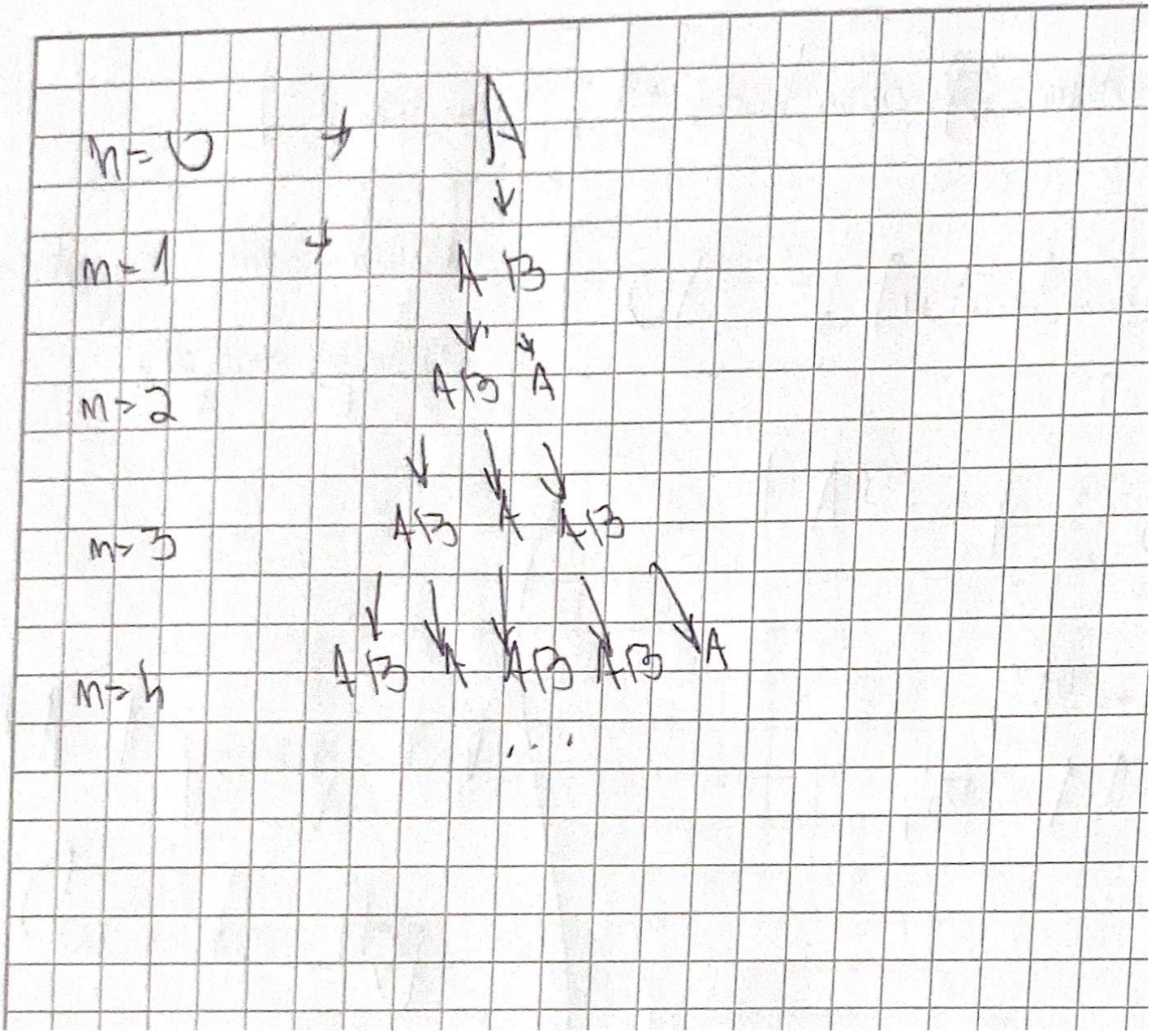
krok pierwszy 1 A→AB

pętla od nowa

krok drugi 2 AB→ ABA

pętla while od nowa (szukamy kolejnego pustego miejsca i aż do jego napotkania zamieniamy i wypisujemy

itd. aż 10 razy


n=0
A
n=1
A B
n=2
A B A
n=3
A B A A B
n=4
A B A B A B A
...