Lab session 4: Imperative Programming

This is the fourth lab of a series of weekly programming labs. You are required to solve these programming exercises and submit your solutions to the automatic assessment system *Themis*. The system is 24/7 online. Note that the weekly deadlines are very strict (they are given for the CET time zone)! If you are a few seconds late, Themis will not accept your submission (which means you failed the exercise). Therefore, you are strongly advised to start working on the weekly labs as soon as they appear. Do not postpone making the labs! You can find Themis via the url `https://themis.housing.rug.nl`

Note: In lab session 4 you are only allowed to make use of constructs that are taught in the weeks 1, 2, 3, and 4 (so no recursion). You are not allowed to use the math library.

Problem 0: What if?

1) Consider the following program:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x;
    scanf("%c", x);
    printf("%i", x);
    return 0;
}
```

Why does this program fail to run correctly?

(a) The variable `x` is not initialized
(b) It assigns a `char` value to `int` variable `x`.
(c) `scanf` expects a pointer rather than an `int`.
(d) It tries to print a `char` value as an `int`.

2) Consider the following line:

```
int x[10] = {1};
```

Which of the following is true after executing this line?

(a) The program has stopped due to a segmentation fault.
(b) The first value of `x` is 1, the remaining 9 values have unknown values.
(c) The first value of `x` is 1, the remaining 9 values have value 0.
(d) All 10 values of `x` are 1.

3) Consider the following function:

```
void doSomething(int *value) {
    *value = 20;
}
```

What happens if we execute `doSomething(&x)` when the value of `x` is initially 30?

(a) This would result in a compiler error.
(b) This would compile, but cause an error at runtime.
(c) The value of `x` will stay 30.
(d) The value of `x` will become 20.

4) Consider the following function:

```
int *getEmptyArray(int size) {
    int newArray[size];
    for (int i = 0; i < size; ++i) {
        newArray[i] = 0;
    }
    return newArray;
}
```

Why does the following function fail to work as expected?

(a) It returns an `int` array when the function declaration says the output is an `int` pointer.
(b) It returns a pointer to a local variable that becomes invalid when the function ends.
(c) The variable `i` is increased with `++i`, while it should be increased with `i++`.
(d) The parameter `size` should be an `int` pointer rather than an `int`.

Problem 1: Smallest Divisors

The input for this exercise is an integer n. Consider the following sequence:

$$f(1) = 1, f(2) = 2, f(n) = m, \tag{1}$$

where $n > 2$ and m is the smallest *unused* divisor of the sum of the two previous term. If a divisor is already used, go to the next 'smallest' divisor until one is found. It could happen that all possible divisors are used. In that case $f(n) = n$. An example would be $f(8) = 11$ and is illustrated in the steps below:

```
f(1) = 1 // 1 is used as a divisor
f(2) = 2 // 2 is used as a divisor
f(3) = 3 // f(1) + f(2) = 1 + 2 = 3, smallest divisor not used is 3
f(4) = 5 // f(2) + f(3) = 2 + 3 = 5, smallest divisor not used is 5
f(5) = 4 // f(3) + f(4) = 3 + 5 = 8, smallest divisor not used is 4
f(6) = 9 // f(4) + f(5) = 5 + 4 = 9, smallest divisor not used in 9
f(7) = 13 // f(5) + f(6) = 4 + 9 = 13, smallest divisor not used is 13
f(8) = 11 // f(6) + f(7) = 9 + 13 = 22, smallest divisor not used is 11.
```

Write a program that reads a single integer ($1 \leq n \leq 10000$) and outputs the value of $f(n)$ according to the description above. Note that 187554 is the largest output possible when adding two subsequent terms in the sequence given the specified range.

Example 1:

input:
```
3
```
output:
```
3
```

Example 2:

input:
```
5
```
output:
```
4
```

Example 3:

input:
```
8
```
output:
```
11
```

Problem 2: 1-D Game of Life

In this problem you implement a version of Game of Life (GoL) yourself. In the lecture slides about arrays, the traditional GoL implementation was discussed which operates on a two dimensional grid. However in this version we make use of a one-dimensional grid in which the simulation is evolved using simple rules:

1. a cell X has 4 neighbouring cells Y and can be represented as `YYXYY`
2. a *living* cell is represented with a `1` and a *dead* cell with a `0`
3. a *dead* cell becomes alive if `2` or `3` neighbour cells are alive
4. a *living* cell survives if `2` or `4` neighbour cells are are alive
5. the representation is *circular*; the first cell is considered to be a neighbour of the last cell

Consider the initial representation `[0 0 0 1 1 0 1 1 0 0 0]`. Evolving this representation over 10 time steps leads to the following (intermediate) representations:

1. Generation 1: `[0 0 1 0 1 0 1 0 1 0 0]`
2. Generation 2: `[0 0 0 1 1 1 1 1 0 0 0]`
3. Generation 3: `[0 0 1 1 0 1 0 1 1 0 0]`
4. Generation 4: `[0 1 0 1 1 1 1 1 0 1 0]`
5. Generation 5: `[1 0 1 0 0 1 0 0 1 0 1]`
6. Generation 6: `[1 1 0 1 1 0 1 1 0 1 1]`
7. Generation 7: `[0 0 0 1 1 0 1 1 0 0 0]`
8. Generation 8: `[0 0 1 0 1 0 1 0 1 0 0]`
9. Generation 9: `[0 0 0 1 1 1 1 1 0 0 0]`
10. Generation 10: `[0 0 1 1 0 1 0 1 1 0 0]`

Write a program that reads from the input two lines. The first line holds two integers $m >= 5$ and n which represent the size of the one-dimensional grid m together with the number of discrete simulation steps n allowing the cells to evolve. On the output you need to print the m-dimensional array holding the current representation after running the simulation for n steps. Regarding the output, there should not be a space before the newline, just a newline.

Example 1:

input:
```
11 10
0 0 0 1 1 0 1 1 0 0 0
```
output:
```
0 0 1 1 0 1 0 1 1 0 0
```

Example 2:

input:
```
5 1
1 1 0 0 1
```
output:
```
1 1 1 1 1
```

Example 3:

input:
```
6 5
1 1 0 0 1 1
```
output:
```
0 1 1 1 1 0
```

Problem 3: Airports

During a summer break you would like to visit a number of cities. You decide to travel by plane and would like to know if it is possible to land in each of these cities in your preferred order. Luckily you know that each city you intend to visit has its own airport. Each airport has one or multiple runways and you may assume that each runway only has one destination.

Write a program that accepts the following from the input. First one line holding three integers n, m and o which represents the number of airports, the maximum number of runways possible per airport and the total number of airports you intend to visit. Then n lines with m integers are listed on the input that represent runways to other cities. These integers should be read as the indices for the different destinations. Note that the order of these n lines is important, as such the value `0` refers to first destination. Note that some airports might have fewer runways, if this is the case non-existing runways are indicated with a `-1`. Finally the input is terminated with a line that denotes the order in which you would like to travel to each city. Take for example the input of the first example:

The input lists `5` airports with a maximum of `3` runways and you visit a total of `4` airports. You want to visit city `0`, `1`, `2` and `0` in that specific order (see last line). The first airport has a runway with destinations to airport `1`, `2` and `3`. Once arrived at airport `1`, you have runways with destinations to airport `2` and `3`. When arriving at airport `2`, it is possible to return again to airport `0`. On the output you should print `YES` if it is possible to travel to each city in the specified order. Otherwise print `NO`.

Example 1:

input:

```
5 3 4
1 2 3
2 3 -1
0 1 -1
0 1 2
1 2 -1
0 1 2 0
```

output:

```
YES
```

Example 2:

input:

```
8 4 7
1 3 2 4
2 3 5 4
3 4 5 6
4 5 6 7
3 5 6 7
3 4 6 7
0 -1 -1 -1
-1 -1 -1 0
1 2 3 4 5 6 0
```

output:

```
YES
```

Example 3:

input:

```
8 4 7
1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
3 5 6 7
3 4 6 7
0 -1 -1 -1
-1 -1 -1 0
1 2 3 4 5 6 7
```

output:

```
NO
```

Problem 4: Rotating Rings

The input of this program is a `n x n` two-dimensional grid consisting of integers. In this program you are asked to rotate some numbers on a ring. A ring is specified by a position `(x, y)` which is the top-left coordinate of the ring, a size `s` and a rotation direction `r`. A rotation means that you move each value one position on the ring either in a clockwise or counterclockwise direction. Note that the top-left coordinate of the grid is `(0, 0)` and the bottom-right coordinate is `(n - 1, n - 1)`. An example clockwise rotation at position `(1, 1)` with size `3` is depicted in the figure below.

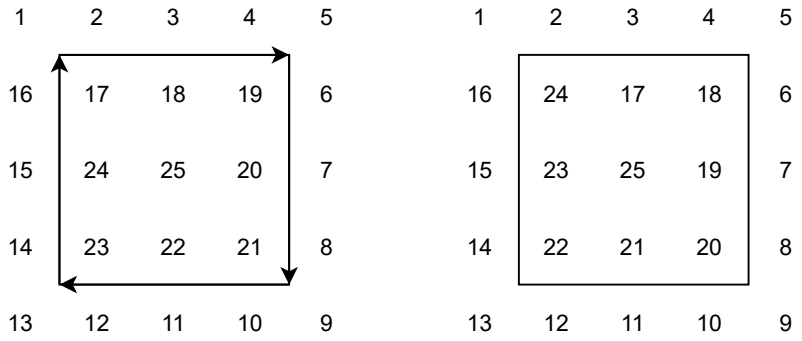

Figure 1: Shifting all values by one in a clockwise fashion.

Write a program that reads several lines from the input. The first line holds a single integer `n` which specifies the size of the `n x n` matrix. The following `n` lines represents the grid. The next m lines represent zero or more rings which are specified with an `(x, y)`-coordinate, size `s` $\geq$ `2` and a direction `r`. The coordinates and sizes are integers, while the direction is specified by either a + for a clockwise motion or a − for a counterclockwise motion. The input is terminated with the value `-1`. Example one corresponds to the figure above. On the output you should print the resulting grid after performing the operations specified by the rings. Regarding the output, there should not be a space before each newline, just a newline.

Example 1:

input:

```
5
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
1 1 3 +
-1
```

output:

```
1  2  3  4  5
16 24 17 18 6
15 23 25 19 7
14 22 21 20 8
13 12 11 10 9
```

Example 2:

input:

```
5
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
0 0 5 +
1 1 4 -
-1
```

output:

```
16 1  2  3  4
15 18 19 5  6
14 17 25 20 7
13 24 22 21 8
12 23 11 10 9
```

Example 3:

input:

```
5
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
1 2 2 -
-1
```

output:

```
1  2  3  4  5
16 17 18 19 6
15 25 22 20 7
14 24 23 21 8
13 12 11 10 9
```