Project 8

This assignment has two parts:

- Part A (`100 pts`)
- Part B (`+100 pts`) (Bonus)
- Beware of the fact that this is the last assignment of the course. Grade weight is **not** twice as much as of a regular assignment unless you implement the bonus part (part B).

Part A:

You are going to write a complete C program which implements the following functionality:

- Your program creates a representation for an `[n x n x n]` Rubik's Cube.
- your program reads a file:
  - `commands.txt`
- According to the information in `commands.txt`, your program creates the necessary structures for the representation and modifies the state of the Rubik's Cube.
- Your program saves the final state of the Rubik's Cube to a file named:
  - `result.txt`
- The initial state of the cube is described below (read the document till the end)
- Rubik's Cube representation and other conventions are given below.

Part B:

Extend the program in Part A so that it supports an additional functionality:

- Shrink. (You are not allowed to reallocate space for this functionality.)
- With this command, the Rubik's Cube shrinks to an `[n-1 x n-1 x n-1]` Rubik's Cube.
- Rubik's Cube representation and other conventions are given below.
- Part B is an extension to part A. Which means, everything written in part A should be applied.

Commands:

Rotation (common for Part A and Part B) (see Figure 1)

- Rows rotate right or left. Columns rotate up or down. A 'rotate' command starts with the keyword `rotate` and has the following format:

```
rotate <Face id> row/column <row/column id> up/down/right/left
```

Examples:

```
rotate 4 row 3 right
```

- This rotates 3rd row of the 4th face right.

```
rotate 3 column 4 down
```

- This rotates 4th column of the 3rd face down.

Shrink (only for Part B) (see Figure 2)

- The cube shrinks to a smaller cube. This means, you have to delete some of the slices. And you have to preserve the rest of them. You are not allowed to allocate space for another cube and copy data for the slices.

`shrink`

- With this command, each `[n x n]` face of your cube shrinks down to `[n-1 x n-1]` by deleting row `0` and column `0`.

`commands.txt` for Part A

- Starts with the dimension information. This follows a list of moves. Each line is a move.
- Example:

```
100
rotate 4 row 4 right
rotate 5 row 20 left
rotate 0 column 75 up
rotate 0 column 12 down
```

- After reading this file, you have to create a `[100 by 100 by 100]` cube and execute the listed moves.
- You don't have to check for any errors.

`commands.txt` for Part B

- Starts with the dimension information. This follows a list of moves. Each line is a move.
- Example:

```
100
rotate 4 row 4 right
rotate 5 row 20 left
shrink
rotate 0 column 75 up
rotate 0 column 12 down
shrink
rotate 2 column 11 up
```

- After reading this file, you have to create a `[100 by 100 by 100]` cube and execute the listed moves. There are 2 `shrink` commands. `shrink` commands can follow other commands. There is no restriction.
- You don't have to check for any errors.

`result.txt` (for parts A and B)

- The final state of the cube should be printed to `result.txt`
- Starting from `face 0`, contents of each face should be listed as a matrix.
- There should be a blank line after each face

Example:

- This is the initial state of a `[3x3x3]` cube.

```
0 0 0
0 0 0
0 0 0

1 1 1
1 1 1
1 1 1

2 2 2
2 2 2
2 2 2

3 3 3
3 3 3
3 3 3

4 4 4
4 4 4
4 4 4

5 5 5
5 5 5
5 5 5
```

- After the following command:

```
rotate 0 row 0 right
```

- The state of the cube is as follows:

```
3 3 3
0 0 0
0 0 0

0 0 0
1 1 1
1 1 1

1 1 1
2 2 2
2 2 2

2 2 2
3 3 3
3 3 3

4 4 4
4 4 4
4 4 4

5 5 5
5 5 5
5 5 5
```

- (For part B only) A `shrink` command modifies the cube as follows:

```
0 0
0 0
```

```
1 1
1 1

2 2
2 2

3 3
3 3

4 4
4 4

5 5
5 5
```

Rubik's Cube Representation

Different sources on World-wide-web extensively covers Rubik's cube.

A Rubik's cube has `6` faces. Each face is divided into slices. For this assignment, slices of your Rubik's cube will hold numbers. **Initially**:

- All of the slices in `Face 0` hold the value `0`
- All of the slices in `Face 1` hold the value `1`
- All of the slices in `Face 2` hold the value `2`
- All of the slices in `Face 3` hold the value `3`
- All of the slices in `Face 4` hold the value `4`
- All of the slices in `Face 5` hold the value `5`

Turn in:

Do not return two different source codes for part A and part B. At the beginning of your source code, **specify** the one you are addressing. **Name** the source code according to your choice.

If you are submitting for **part A**:

A complete C program `<Project8.c>` which can be compiled using the following command:

```
gcc -std=c99 Project8.c -o Project8
```

If you are submitting for **part B**:

A complete C program `<assignment_8_part_B_name_id.c>` which can be compiled using the following command:

```
gcc -std=c99 assignment_8_part_B_name_id.c -o assignment_8_part_B_name_id
```

If your program requires additional compile and link options, state that requirement at beginning of your source code as a comment.

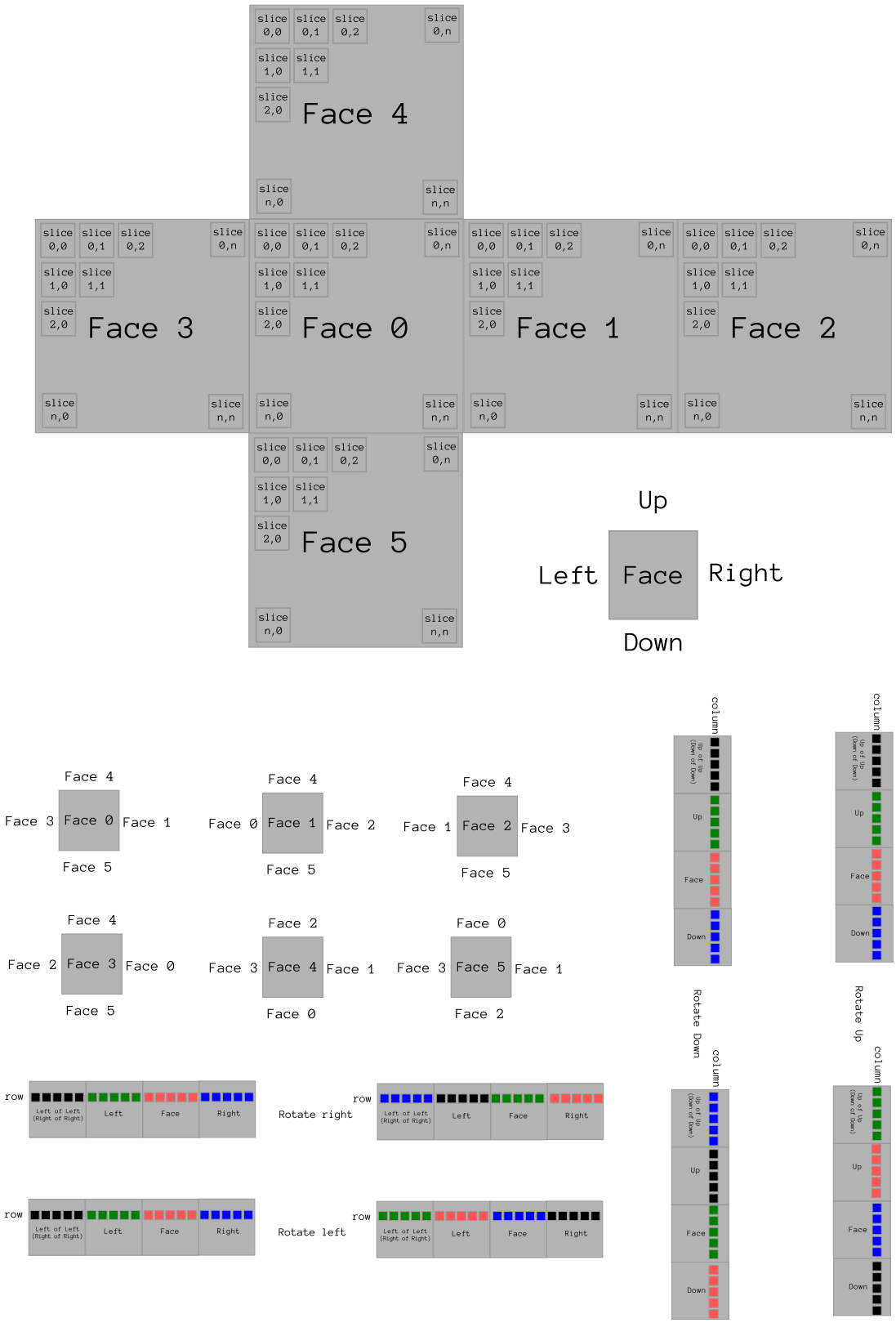

Figure 1: Cube Conventions and Rotate command

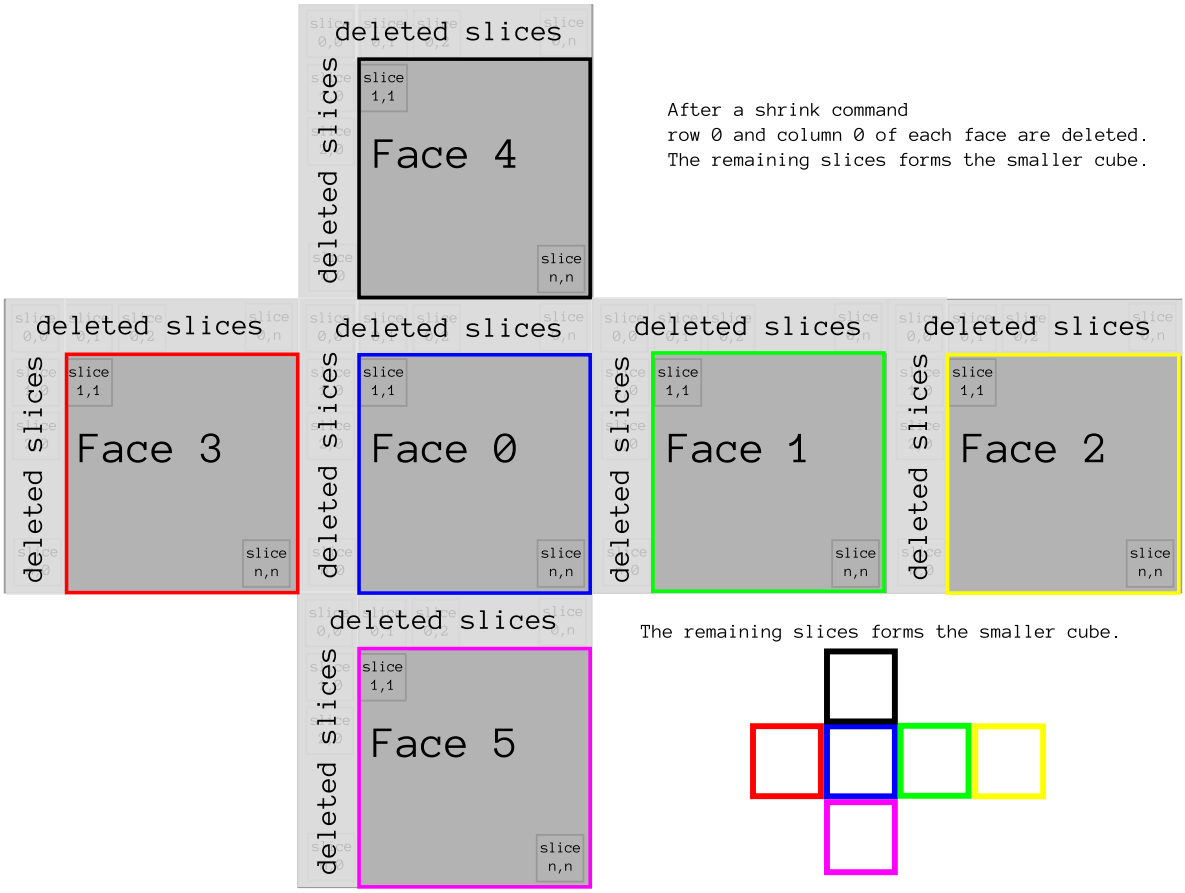

Figure 2: Shrink operation