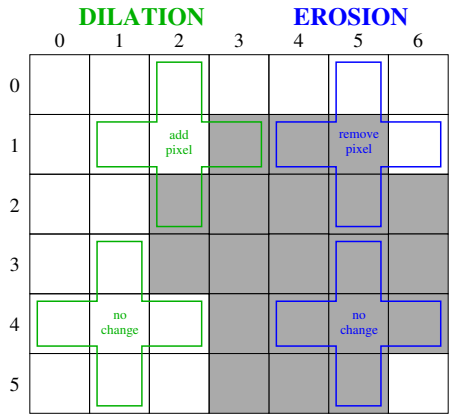
CSCI-1200 Data Structures — Spring 2022
Homework 1 — Morphological Image Processing

Before starting this homework, make sure you have read the Collaboration Policy & Academic Integrity.

At the core of many advanced computer vision algorithms for image segmentation, feature detection, and classification are several basic morphological image processing operations that modify the borders between neighboring clusters of similarly colored pixels. The *dilation* operation adds pixels along the boundary to expand or grow a cluster. The *erosion* operation shrinks the boundary of the cluster by reassigning pixels along the boundary to a background color. We'll also implement a simple *replace* operation to change all pixels in the image of color A to color B.

The diagram on the right illustrates the details of the dilation and erosion operations. In both cases we will use a plus-shaped *structuring element* that contains the 4 adjacent pixels to a candidate pixel. When deciding which pixels to add to expand the boundary of the grey cluster during a *dilation* operation, we place the structuring element over each candidate pixel and ask if the shape overlaps or "hits" any of the existing grey pixels. If it does, we change its color to the foreground color (grey). In contrast, when shrinking the boundary during *erosion*, we consider removing each pixel from the cluster. If all of the pixels of the structuring element do not fully "fit" within the grey cluster of pixels we will set that pixel to the background color (white).



You will work with command line arguments, file input and output, and the STL C++ `string` and `vector` classes to manipulate manipulate a grid of ASCII characters (instead of a traditional color image file). Please read the entire assignment carefully before starting to program.

Command Line Arguments

Your program will expect 4-5 command line arguments, depending on the requested operation. The 1st argument is the name of the input ASCII art image file. The 2nd argument is the name of the output file where the ASCII art image resulting from the operation should be written. The 3rd argument is the name of one of 3 operations that should be performed: `replace`, `dilation`, or `erosion`. Here are some example command lines:

```
./image_processing.out input4.txt output4_replace.txt replace X O
./image_processing.out input4.txt output4_dilation.txt dilation X
./image_processing.out input4.txt output4_erosion.txt erosion X \.
```

The number and purpose of the remaining arguments on the command line depend on the operation. For a *replace* operation, the 4th argument is the character that should be removed from the image and the 5th argument is the character that should be substituted in those locations. For the *dilation* operation, the 4th argument is the foreground character (clusters of pixels of this "color" should increase in size). For the *erosion* operation, the 4th argument is the foreground character (clusters of pixels of this "color" should decrease in size). We also need to know the background color that will replace pixels that are removed from the foreground object and the 5th argument is that background character.

Note that certain ASCII characters have special meaning on the command line, and it will be necessary to *escape* that meaning by preceding those characters by a backslash (\) to make sure your program receives the desired character.

Examples and More Information

In many computer vision applications, these operations are applied sequentially, one after the other. Note that these operations are generally not commutative. The *opening* operation is the result of first applying erosion and then dilation. The effect of opening is to round sharp outer corners and remove tiny islands. The *closing* operation is achieved by first performing dilation and then erosion. The effect of closing is to fill in small gaps within the foreground object and smooth inner corners.

```
./image_processing.out output4_erosion.txt output4_opening.txt dilation X
./image_processing.out output4_dilation.txt output4_closing.txt erosion X \.
```

```
    input4.txt       output4_replace.txt   output4_dilation.txt   output4_erosion.txt   output4_opening.txt   output4_closing.txt
..............   ..............   ..............   ..............   ..............   ..............
..............   ..............   ..XXX.....X..   ..............   ..............   ..............
..XXX.....X..   ..OOO.....O..   .XXXXX...XXX.   ..............   ...X..........   ..XXX.....X..
..XXX.....X..   ..OOO.....O..   .XXXXX...XXX.   ...X..........   ..XXX........   ..XXX.....X..
..XXX........   ..OOO........   .XXXXX....X..   ...X..........   ..XXX........   ..XXX........
..XXX........   ..OOO........   .XXXXXXXX....   ...X..........   ..XXX........   ..XXXX.......
..XXXXXXX....   ..OOOOOOO....   .XXXXXXXXX...   ...XX.........   ..XXXXXX.....   ..XXXXXXX....
..XXXXXXX....   ..OOOOOOO....   .XXXXXXXXX...   ...XXXXX......   ..XXXXXXX....   ..XXXXXXX....
..XXXXXXX....   ..OOOOOOO....   .XXXXXXXXX...   ..............   ...XXXXX.....   ..XXXXXXX....
..............   ..............   ..XXXXXXX....   ..............   ..............   ..............
..............   ..............   ..............   ..............   ..............   ..............
```

You must follow the specifications for the command line arguments and functionality exactly as described in this handout to receive full credit. Sample input and output files are posted on the course website, and the validation and automatic grading on the submission server will also help you check your work. More information on command line arguments and file I/O in C++ is available on the course webpage under "Misc. C++ Programming Info".

Image Bounds & Other Error Checking

When the structuring element is placed at the edge of the image, one or more pixels may fall off the edge and not overlap legal image coordinates. You do not want to access data that does not exist (a memory error that could give weird output or crash the program!). Be sure to add appropriate checks in your program to skip these out-of-bounds coordinates.

You may assume that the input image file is properly formatted and contains a rectangular grid of non-white space characters. However, to help with your own debugging, you should implement simple error checking to ensure that the command line arguments are appropriate and that the input and output file streams are successfully opened. Your program should exit gracefully with a useful error message sent to `std::cerr` if there is a problem with the arguments.

Submission Details

Do your work in a folder named `hw1` within your Data Structures homeworks directory. Organize your program into logical, reasonable-length functions – don't put all of the code in `main`! Be sure to read the "Homework Grading Criteria" as you put the finishing touches on your solution. Create new test cases to fully debug your program and don't forget to comment your code! Use the provided template `README.txt` file for notes you want the grader to read. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.