Intermediate Programming

Day 16

Outline

- Exercise 16
- Midterm project

Exercise 16

Implement **length**.

list.h

```
…
unsigned int length( const Node *head );
…
```

list.c

```
…
unsigned int length( const Node *head )
{
	unsigned int len = 0;
	while( head ) len++ , head = head->next;
	return len;
}
…
```

list.c

```
…
unsigned int length( const Node *head )
{
	unsigned int len = 0;
	for( ; head ; head=head->next , len++ ) ;
	return len;
}
…
```

Exercise 16

list.h

```
…
unsigned int length( const Node *head );
…
```

list.c

```
…
unsigned int length( const Node *head )
{
	if( !head ) return 0;
	else return 1 + length( head->next );
}
…
```

Implement `length` recursively.

Base Case:

- The linked list has no elements.

Recursion:

- The length of the whole list is one plus the length of the sub-list linked off of the `next` pointer.

Exercise 16

Implement add_after.

list.h

```
…
add_after( Node *n , char c );
…
```

list.c

```
…
int add_after( Node *n , char c )
{
	Node *_n = create_node(c);
	if( !_n ) return 1;
	_n->next = n->next;
	n->next = _n;
	return 0;
}
…
```

Exercise 16

Implement **reverse_print** recursively.

Base Case:

- If the list has one elements, print its contents.

Recursion:

- Reverse print the sub-list linked off of the **next** pointer.
- Then print **node**'s contents

list.h

```
…
void reverse_print( const Node *node );
…
```

list.c

```
…
void reverse_print( const Node *node )
{
	if( !node->next ) printf( "%c " , node->data );
	else
	{
		reverse_print( node->next );
		printf( "%c " , node->data );
	}
}
…
```

Exercise 16

Implement **reverse_print** recursively.

list.h

```
…
void reverse_print( const Node *node );
…
```

list.c

```
…
void reverse_print( const Node *node )
{
	if( node->next ) reverse_print( node->next );
	printf( "%c " , node->data );
}
…
```

Base Case:

- If the list has one elements, print its contents.

Recursion:

- Reverse print the sub-list linked off of the **next** pointer.
- Then print **node**’s contents

Outline

- Exercise 16
- Midterm project

Midterm project

Image Representation:

An image has a prescribed number of rows (height) and columns (width), as well as a list of pixel values.

ppm_io.h

```
...
typedef struct {
      unsigned char r;
      unsigned char g;
      unsigned char b;
} Pixel;

typedef struct {
      Pixel *data;
      int rows;
      int cols;
} Image;
...
```

Midterm project

Image Representation:

An image has a prescribed number of rows (height) and columns (width), as well as a list of pixel values.

A pixel is described by its red, green, and blue values.

ppm_io.h

```
…
typedef struct {
      unsigned char r;
      unsigned char g;
      unsigned char b;
} Pixel;

typedef struct {
      Pixel *data;
      int rows;
      int cols;
} Image;
…
```

Midterm project

ppm_io.h

```
…
typedef struct {
	Pixel *data;
	int rows;
	int cols;
} Image;
…
Image make_image( int rows , int cols );
void free_image( Image * im );
int write_ppm( FILE * fp , const Image img );
Image read_ppm( FILE * fp );
…
```

Image Storage:

Images are read from / written to disk using the `.ppm` file format.
(See the midterm webpage for details.)

You will need to implement functionality for:

Midterm project

Image Storage:

Images are read from / written to disk using the `.ppm` file format.
(See the midterm webpage for details.)

ppm_io.h

```
…
typedef struct {
      Pixel *data;
      int rows;
      int cols;
} Image;
…
Image make_image( int rows , int cols );
void free_image( Image * im );
int write_ppm( FILE * fp , const Image img );
Image read_ppm( FILE * fp );
…
```

You will need to implement functionality for:

- Creating an `Image` of the prescribed width/height (and allocating the pixels)

Midterm project

Image Storage:

Images are read from / written to disk using the `.ppm` file format.
(See the midterm webpage for details.)

ppm_io.h

```
...
typedef struct {
      Pixel *data;
      int rows;
      int cols;
} Image;
...
Image make_image( int rows , int cols );
void free_image( Image * im );
int write_ppm( FILE * fp , const Image img );
Image read_ppm( FILE * fp );
...
```

You will need to implement functionality for:

- Creating an `Image` of the prescribed width/height (and allocating the pixels)
- Deallocating the pixels of an `Image` (and setting the `data` member to `NULL`)

Midterm project

ppm_io.h

```
…
typedef struct {
	Pixel *data;
	int rows;
	int cols;
} Image;
…
Image make_image( int rows , int cols );
void free_image( Image * im );
int write_ppm( FILE * fp , const Image img );
Image read_ppm( FILE * fp );
…
```

Image Storage:

Images are read from / written to disk using the `.ppm` file format.
(See the midterm webpage for details.)

You will need to implement functionality for:

- Creating an `Image` of the prescribed width/height (and allocating the pixels)
- Deallocating the pixels of an `Image` (and setting the `data` member to `NULL`)
- Writing an `Image` to a (binary) file handle

Midterm project

ppm_io.h

```
…
typedef struct {
	Pixel *data;
	int rows;
	int cols;
} Image;
…
Image make_image( int rows , int cols );
void free_image( Image * im );
int write_ppm( FILE * fp , const Image img );
Image read_ppm( FILE * fp );
…
```

Image Storage:

Images are read from / written to disk using the `.ppm` file format.
(See the midterm webpage for details.)

You will need to implement functionality for:

- Creating an `Image` of the prescribed width/height (and allocating the pixels)
- Deallocating the pixels of an `Image` (and setting the `data` member to `NULL`)
- Writing an `Image` to a (binary) file handle

Note:

Functionality for reading an `Image` is provided, but it requires `make_image` being (correctly) implemented.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1, const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Grayscale:

Given an input image, return an image whose pixels are the grayscale values of the input pixels.

- An output pixel is gray, if its red, green, and blue values are equal
- Given red (r), green (g), and blue (b) input pixel values, the corresponding gray value is given by:

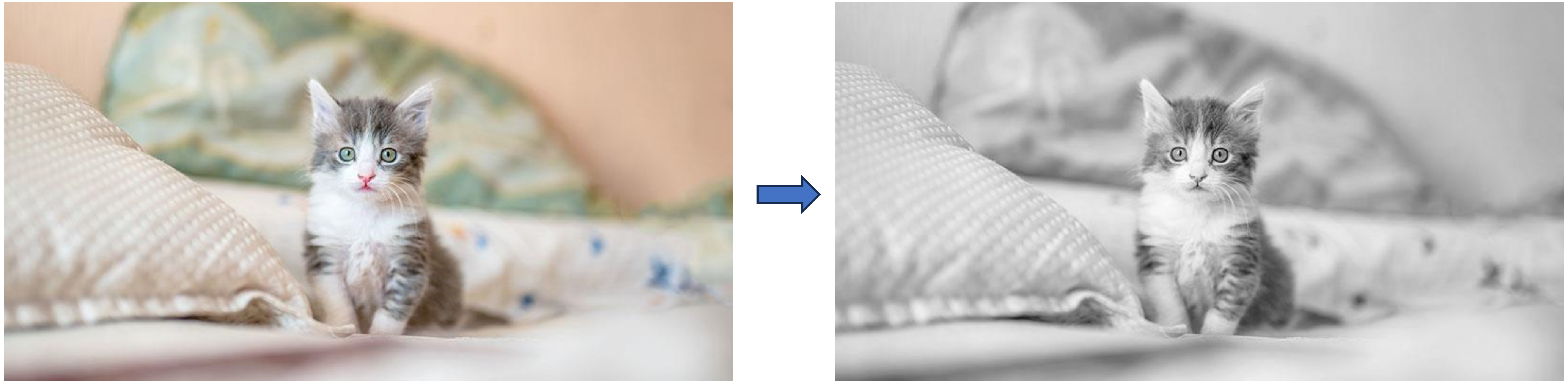
$$\text{Gray}(r, g, b) = 0.3r + 0.59g + 0.11b$$

Midterm project

image_manip.h

```
...
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
...
```

<u>Grayscale</u>:



Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Blend ($0 \leq \alpha \leq 1$):

Given a source and target image, return an image:

- Whose width/height is the maximum of the widths/heights of the inputs
- Whose pixel values are the α-blend of the pixel values of the input.
  For example:
    - If the red channels of the inputs are r_1 and r_2, the red channel of the output will be:
      $$\alpha \cdot r_1 + (1 - \alpha) \cdot r_2$$
    - If only one of the pixels is defined, use that pixel's value.

Midterm project

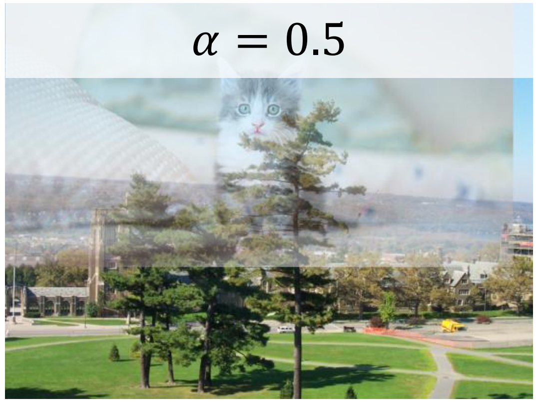
image_manip.h

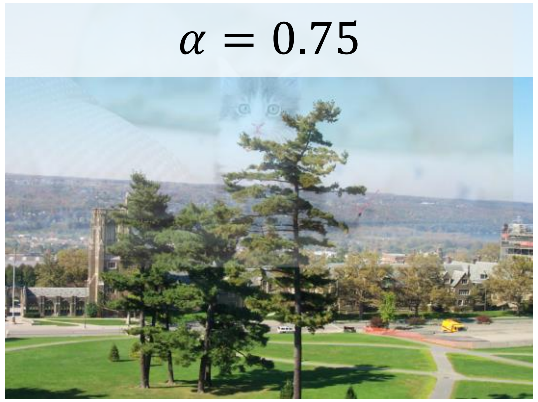
```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

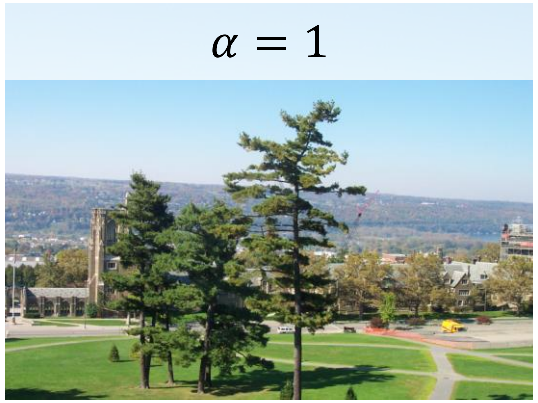
Blend ($0 \leq \alpha \leq 1$):


$\alpha = 0$


$\alpha = 0.25$


$\alpha = 0.5$


$\alpha = 0.75$


$\alpha = 1$

Midterm project

image_manip.h

```
...
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
...
```

<u>Rotate-CCW</u>:

Rotate the image counter-clockwise by 90°.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

<u>Pointilism</u>:

Return the result of applying a pointillist filter to the input.

For a random subset of the pixels in the image:

- Draw a filled-in circle at the pixel, with the pixel’s color
    - The radius of the circle should itself be a random value between 1 and 5.

<u>[WARNING]</u>

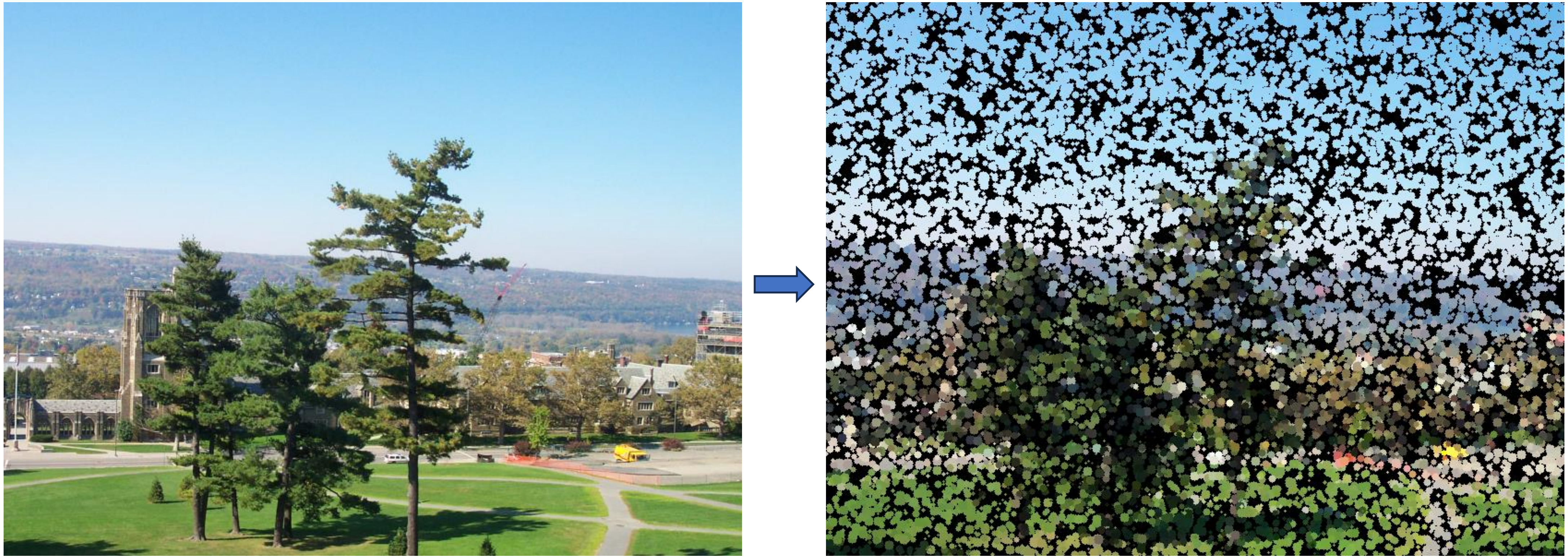
- Because you will be using random values, your results may not be identical to those you see on the webpage.
- For reproducibility, `pointilism` takes a seed for random number generation.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

<u>Pointilism</u>:



Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Blur ($\sigma > 0$):

Return the result of smoothing the input.

- For each output pixel, compute the weighted average of the nearby pixels in the input.
  - The value of the weight should only depend on the distance of the nearby pixels.
    - The weight should be non-negative
    - The weight should fall off with distance
    - The sum of the weight should be one.

  ⇒ You will set the weights using a Gaussian stencil with the size of the stencil (and standard deviation of the Gaussian) a command-line parameter.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Blurring Stencil:

Given a standard deviation σ you will define a $(2r_\sigma + 1) \times (2r_\sigma + 1)$ grid of filtering values, F, where:

- The radius is roughly five times the standard deviation:

$$r_\sigma \approx 5\sigma$$

- The value in the (i, j)-th entry of the grid is given by the normal distribution:[*]

$$F[i][j] \sim \exp\left(-\frac{(i - r_\sigma)^2 + (j - r_\sigma)^2}{2\sigma^2}\right)$$

---

[*] Recall that the center is at $(i = r_\sigma, j = r_\sigma)$ which is where the filter is largest.

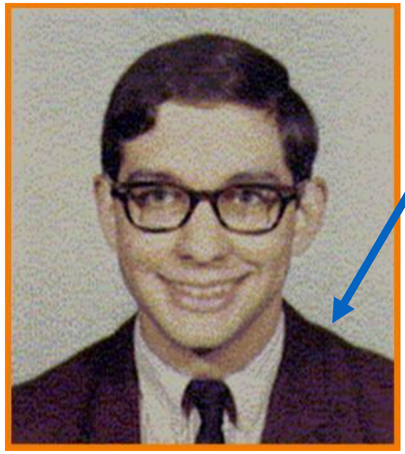
Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Blurring Stencil:

Pixel(x,y): red = 36
green = 36
blue = 0



Original

$$F = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

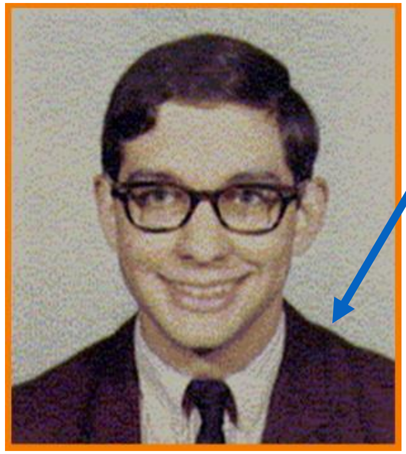
Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Blurring Stencil:

Pixel(x,y): red = 36
green = 36
blue = 0



Original

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$F = \begin{bmatrix} {}^{1}/_{16} & {}^{2}/_{16} & {}^{1}/_{16} \\ {}^{2}/_{16} & {}^{4}/_{16} & {}^{2}/_{16} \\ {}^{1}/_{16} & {}^{2}/_{16} & {}^{1}/_{16} \end{bmatrix}$$

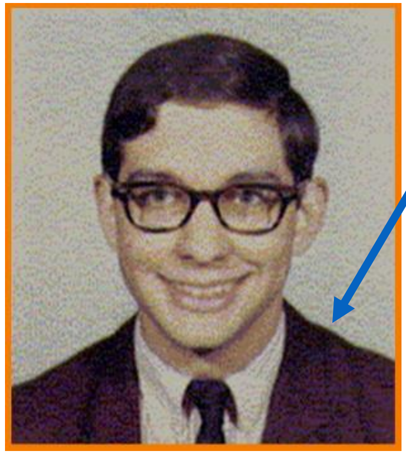
Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

<u>Blurring Stencil</u>:

New value for Pixel(x,y).red =
(36 * 1/16) + (109 * 2/16) + (146 * 1/16)
(32 * 2/16) + (36 * 4/16) + (109 * 2/16)
(32 * 1/16) + (36 * 2/16) + (73 * 1/16)



Original

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

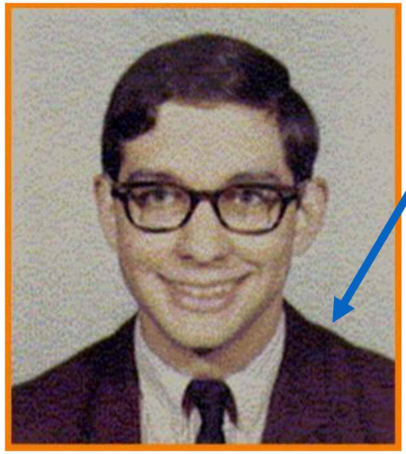
$$F = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Midterm project

```
image_manip.h
...
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
...
```

Blurring Stencil:

New value for Pixel(x,y).red = 62.69



Original

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$F = \begin{bmatrix} {}^{1}/_{16} & {}^{2}/_{16} & {}^{1}/_{16} \\ {}^{2}/_{16} & {}^{4}/_{16} & {}^{2}/_{16} \\ {}^{1}/_{16} & {}^{2}/_{16} & {}^{1}/_{16} \end{bmatrix}$$

Midterm project

image_manip.h

```
...
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
...
```

Blur ($\sigma > 0$):

$\sigma = 1$

$\sigma = 5$

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Saturate ($s \geq 0$):

Return the result of increasing/decreasing the saturation of the pixels in the input by a factor of s.

Definition:

The saturation is the extent to which the color deviates from its grayscale value.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

Saturate ($s \geq 0$):

Return the result of increasing/decreasing the saturation of the pixels in the input by a factor of s.

For each pixel:

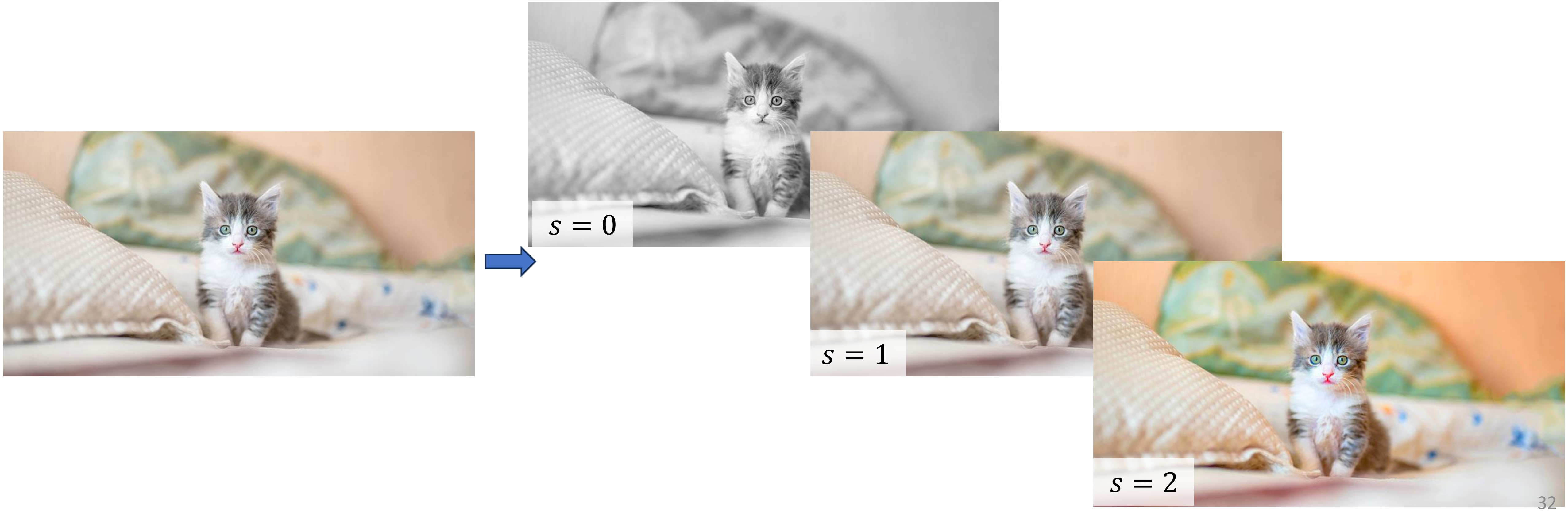
- Compute its grayscale value
- Compute the difference between the pixel's value and the grayscale value.
- Scale the difference by s and add that back to the grayscale.

Midterm project

image_manip.h

```
…
Image grayscale( const Image in );
Image blend( const Image in1 , const Image in2 , double alpha );
Image rotate_ccw( const Image in );
Image pointilism( const Image in , unsigned int seed );
Image blur( const Image in , double sigma );
Image saturate( const Image in , double scale );
…
```

<u>Saturate ($s \geq 0$)</u>:



Midterm project

Things to keep in mind:

- The pixels of an image are standardly stored in row-major format:
  - All the pixels of the first row are stored before the pixels of the second, which are stored before the pixels of the third, etc.
  - Within a row, pixels are ordered by column.
- The $(0,0)$ pixel is at the top left of the image.
- All procesing returns a new `Image`. (Input should not be modified.)
- The red, green, and blue values are stored as `unsigned char`s but many of the applications require doing calculations with floating point precision:
  - Be aware of implicit casting
  - Be careful when floating point values are outside the range that can be represented by an `unsigned char`.
- When applying a (smoothing) filter, make sure that you don't try accessing neighboring pixels that are not in the image

Midterm project

Things to keep in mind:

- The pixels of an image are standardly stored in row-major format:
  - All the pixels of the first row are stored before the pixels of the second, which are stored before the pixels of the third, etc.
  - Within a row, pixels are ordered by column.
- The $(0,0)$ pixel is at the top left of the image.
- All procesing returns a new `Image`. (Input should not be modified.)
- The red, green, and blue values are stored as `unsigned char`s but many of the applications require doing calculations with floating point precision:
  - Be aware of implicit casting
  - Be careful when floating point values are outside the range that can be represented by an `unsigned char`.
- When applyin n’t try accessing
  neighboring p

`valgrind` is your friend!