TDT4102 – August 2018, løsningsforslag.

All kode er testet og kjørt. Dette er én av mange mulige løsninger som ville ha gitt maksimal uttelling på eksamen. Vi ber om forståelse for at vi ikke har kapasitet til å lage et LF som viser mange ulike løsningsmuligheter på de ulike delspørsmål. (Husk at alle ikke-trivielle programmeringsoppgaver kan løses på utallige måter.)

Sist oppdatert: 25/8-2018, *Lasse Natvig*

Oppgave 1: Kodeforståelse

1a) **21 21 3 2**

1b) **-196 3 1**

1c) **abcd123 7 c cd**

1d) **6**

1e) **5 Exception: Denominator is 0**

1f) **11.1 22.2 33.3**

1g) **found 1.5**

1h) **3 3 4**

Oppgave 2:

2a)

```
int main() {
	ifstream inFile;
	inFile.open("input.txt");  // mer kompakte løsninger finnes og kan også gi full score
	if (inFile.fail()) {
		cout << "Error opening file!" << endl;
		exit(1);
	}
	int x;
	vector<Obs> all;
	inFile >> x;
	while (x >= 0) {
		Obs next;
		next.x = x;
		inFile >> next.y;
		all.push_back(next);
		inFile >> x;
	}
	inFile.close();
	return 0;
}
```

2b)

```
bool operator<(Obs lhs, Obs rhs) {
	return ((lhs.x < rhs.x) ||
		((lhs.x == rhs.x) && lhs.y < rhs.y));
}
```

2c)

```
void report(vector<Obs> vec, int threshold) {
	map<Obs, int> count;
	for (auto o : vec) {
		count[o]++;
	}
	for (auto c : count) {
		if (c.second > threshold) {
			cout << c.first.x << " " << c.first.y << " " << c.second << endl;
		}
	}
}
```

Oppgave 3:

3a)

```
float clip(float n, float lower, float upper) {
	return max(lower, min(n, upper));
}
```

3b) Siden r, g, og b alle er i området 0..255 er det hensiktsmessig å bruke uint8_t slik at en ikke bruker unødig mye plass i minnet. Unsigned char vil også fungere.

```
// static_cast needed to avoid warning, not required by student
struct Color {
	uint8_t r;
	uint8_t g;
	uint8_t b;
	Color() : r(0), g(0), b(0) {}
	Color(int red, int green, int blue) {
		r = static_cast<uint8_t>(clip(static_cast<float>(red), 0, 255));
		g = static_cast<uint8_t>(clip(static_cast<float>(green), 0, 255));
		b = static_cast<uint8_t>(clip(static_cast<float>(blue), 0, 255));
	}
};
```

3c)

```
Image::Image(int width, int height) : width(width), height(height) {
	data = new Color[width * height];
}
```

3d)

```
Image::Image(const Image &other) : width(other.width), height(other.height) {
	data = new Color[width * height];
	for (unsigned int i = 0; i < width * height; i++) {
		data[i] = other.data[i];
	}
}
```

3e)

```
Image::~Image() {
	delete[] data;
}
```

3f) Medlemsvariabelen data peker til dynamisk allokert minne. Hvis vi bruker den implementasjonen av tilordningsoperatoren som kompilatoren automatisk genererer så vil kun pekerverdien kopieres over. Vi vil da ha to objekter som peker til samme minne. For å unngå dette så må vi implementere tilordningsoperatoren som allokerer nytt minne og kopierer over data dit. (Dette er forskjellen på grunn og dyp kopi).

3g)

```
Color Image::getPixel(int x, int y) {
	return data[y * width + x];
}
void Image::setPixel(int x, int y, Color c) {
	data[y * width + x] = c;
}
```

3h)

```
Image Image::grayscale() {
	Image dest(*this);
	for (unsigned int i = 0; i < width * height; i++) {
		Color p = data[i];
		p.r = p.g = p.b = (p.r + p.g + p.b) / 3;
		dest.data[i] = p;
	}
	return dest;
}
```

3i)

```
uint8_t thresh(uint8_t val, uint8_t t) {
	if (val >= t) return 255;
	else return 0;
}
Image Image::threshold(unsigned int t) {
	Image dest(*this);
	for (unsigned int i = 0; i < width * height; i++) {
		Color p = data[i];
		p.r = thresh(p.r, t);
		p.g = thresh(p.g, t);
		p.b = thresh(p.b, t);
		dest.data[i] = p;
	}
	return dest;
}
```

3j)

```
Image Image::operator+(Image other) {
	if (width != other.width || height != other.height) {
		throw invalid_argument("Size mismatch");
	}
	Image dest(width, height);
	for (unsigned  int y = 0; y < height; y++) {
		for (unsigned  int x = 0; x < width; x++) {
			Color p1 = getPixel(x, y);
			Color p2 = other.getPixel(x, y);
			Color p((p1.r + p2.r) / 2, (p1.g + p2.g) / 2, (p1.b + p2.b) / 2);
			dest.setPixel(x, y, p);
		}
	}
	return dest;
}
```

3k) Bruk av assert passer fint her, men kreves ikke av studenten da vi ikke spurte om det. Denne koden gir en del advarsler (warning) som kan unngås ved bruk av static_cast() men vi har ikke tatt det med i LF for enkelhetsskyld og da det ikke kreves i denne deloppgaven.

```
Color Image::applyKernel(int x, int y, Kernel k) {
	float r = 0;
	float g = 0;
	float b = 0;
	assert(x > 0);
	assert(x < width - 1);
	assert(y > 0);
	assert(y < height - 1);
	for (unsigned int j = 0; j < 3; j++) {
		for (unsigned int i = 0; i < 3; i++) {
			Color p = getPixel(x - 1 + i, y - 1 + j);
			r += p.r * k[j][i];
			g += p.g * k[j][i];
			b += p.b * k[j][i];
		}
	}
	return Color(r, g, b);
}
```

3l)

```
Image Image::convolve(Kernel k) {
	Image dest(*this);
	for (unsigned int y = 1; y < height - 1; y++) {
		for (unsigned int x = 1; x < width - 1; x++) {
			Color p = applyKernel(x, y, k);
			dest.setPixel(x, y, p);
		}
	}
	return dest;
}
```

Oppgave 4

4a)

```
class Layer {
private:
	int N;     // Number of nodes
	int M;     // Number of input synapses per node
	vector<double> a; // Output of the N nodes
	vector<double> w; // Weights for the MxN synapses
public:
	Layer(int N, int M);
	int getSize() { return N; }
	int getInputSize() { return M; }
};
```

4b)

```
Layer::Layer(int N, int M) : N(N), M(M) {
	for (int i = 0; i < N; i++) {
		a.push_back(0);
	}
	for (int i = 0; i < N*M; i++) {
		w.push_back(1);
	}
}
```

4c)

```
void Layer::setWeight(int j, int i, double weight) {
	w[j * M + i] = weight;
}
double Layer::getWeight(int j, int i) {
	return w[j * M + i];
}
```

4d)

fn må deklareres virtual for at subklassens versjon blir kalt av funksjoner i Layer.

*) Skjønn

4e)

```
void Layer::forward(const vector<double> &input) {
	if (input.size() != M) {
		throw invalid_argument("Input size mismatch");
	}
	for (int j = 0; j < N; j++) {
		a[j] = 0;
		for (int i = 0; i < M; i++) {
			a[j] += getWeight(j, i) * input[i];
		}
		a[j] = fn(a[j]);
	}
}
```

4f)

```
void Layer::forward(Layer *input) {
	forward(input->a);
}
```

4g)

```
class SigmoidLayer : public Layer {
public:
	SigmoidLayer(int N, int M) : Layer(N, M) { }
	virtual double fn(double x) {
		return 1 / (1 + exp(-x));
	}
};
```

4h)

```
void NeuralNet::addLayer(Layer *layer) {
	if (layers.size() > 0) {
		// Layer *prev = layers.back(); // vector.back() is not part of the course, therefore:
		Layer *prev = layers[layers.size() - 1];
		assert(prev->getSize() == layer->getInputSize());
	}
	layers.push_back(layer);
}
```

4i)

```
Layer *NeuralNet::forward(const vector<double> &input) {
	layers[0]->forward(input);
	for (int i = 1; i < layers.size(); i++) {
		layers[i]->forward(layers[i - 1]);
	}
	return layers[layers.size() - 1];
}
```

…---ooo0ooo---…