Lista 5 – klasy

Na wykładzie poznali państwo klasy w najprostszej ich wersji. Pora nabrać wprawy w „notacji z kropką” oraz programami składającymi się z wielu plików. Proszę zwrócić uwagę, jak klasa definiuje kompletny zbiór operacji do obiektach, które do niej należą.

Załączone na TEAMS pliki `main.cpp`. `image.h`, `image.cpp` zawierają szkielet prostej biblioteki do obróbki zdjęć oraz programu, który ją testuje. Biblioteka wczytuje i zapisuje pliki w formacie `PPM` (http://netpbm.sourceforge.net/doc/ppm.html , https://en.wikipedia.org/wiki/Netpbm) w wersji binarnej („`P6`”). Plik `image.h` zawiera gotowy interfejs klasy `Image` i kilku typów pomocniczych, nie widzę powodu, by cokolwiek w nim zmieniać. Plik `image.cpp` zawiera gotową implementację składowej `load`. I nic więcej. Proszę dostarczyć implementację brakujących 15 funkcji. Te, które zawierają jakąkolwiek pętlę, powinny znaleźć się w pliku `image.cpp`. Te, których implementacja jest krótka, mogą być zaimplementowane w pliki `image.h`. Funkcje te można testować na załączonych plikach `obrazek.ppm` (wiadomo, co przedstawia) i `drzewo.ppm` (https://kirkville.com/how-to-add-sepia-tone-to-photos-in-apple-photos/). Przykładowa lista programów, które otwierają pliki w formacie ppm: https://fileinfo.pl/extension/ppm. Jeżeli używasz Windows i nie masz żadnego programu obsługującego ten format, to IrfanView jest godny polecenia.
Objaśnienie brakujących funkcji (w kolejności alfabetycznej, poza pierwszą i ostatnimi dwiema, najłatwiejszymi):

1. `void save_as(const std::string& filename);`
   Zapisuje obrazek w pliku o podanej nazwie. To jest pierwsza funkcja składowa, którą należy zaimplementować, bo bez niej nie można wykonać testów. Proszę zapoznać się z implementacją funkcji `load`. Funkcję `save_as` można napisać bardzo, bardzo podobnie.
2. `void add_watermark();`
   Dodaje do obrazka „znak wodny”, np. przekreślenie w kształcie krzyżyka, poziomą linię itp. Coś prostego, ale widocznego.
3. `void blurr();`
   Zmiękcza obrazek poprzez zastąpienie wartości składowych kolorów ich średnią (np. arytmetyczną z równymi wagami) z tego piksela i 8 sąsiednich pikseli, graniczących z nim brzegiem lub wierzchołkiem (uwaga na brzegi obrazka!). Do testów warto tę transformację zastosować wiele razy, np. 10 albo nawet 100.
4. `void extract_layer(ColorLayer color_layer);`
   Zamienia obrazek na jedną z trzech warstw odpowiadających palecie RGB (red, green, blue). Czyli zeruje dwie składowe koloru, a trzecią pozostawia bez zmian.
5. `void filter();`
   „Magiczny filtr” polega na zastąpieniu wartości składowych kolorów piksela (`x, y`) wartościami bezwzględnymi różnic tych składowych w pozycji (`x, y`) i (`x+1, y+1`). Uważaj na brzegi. Bardzo ciekawy efekt, zwłaszcza dla zdjęć monochromatycznych (np. dla załączonego zdjęcia drzewa). Można poeksperymentować, zastępując różnicę operatorem bitowym, np. `^` lub `&`.
6. `void flip_horizontally();`
   Odwróć obrazek w poziomie
7. `void flip_vertically();`
   Odwróć obrazek w pionie
8. `void inflate();`
   Powiększ obrazek 2 razy. Tam, gdzie w oryginale jest piksel `(x, y)`, tam w powiększeniu jest grupa 4 identycznych pikseli tworzących kwadrat o wierzchołku `(2x, 2y)`. Nie kombinuj z interpolacją, to nie są zajęcia z grafiki.
9. `void negative();`
   Zastąp obrazek jego negatywem.
10. `void rotate_clockwise_90();`
    Obróć obrazek o 90 stopni w kierunku obrotu wskazówek zegara.
11. `void sepia();`
    Zamień obrazek na odcienie szarości, a następnie nadaj mu odcień sepii. Ja używałem jako ustawienia bazowego sepii wartości znalezionych w internecie: czerwony=162, zielony=128 i niebieski=101. Jak z tak skąpej informacji nadać obrazkowi w odcieniach szarości jakiś z góry zakładany odcień? Łatwiej, niż w tej chwili może się wydawać, u mnie 7 linijek nie licząc wierszy zwierających wyłącznie klamry.
12. `void shrink();`

Operacja odwrotna do `inflate()`, czyli zmniejszamy rozmiar obrazka o czynnik 2. Do piksela `(x.y)` wpisujemy średnią arytmetyczną, w każdym kanale kolorów liczoną osobno, z wartości koloru w czterech sąsiednich pikselach oryginału. Jak obliczyć ich współrzędne?

13. `void to_grayscale();`

    Zamień (kolorowy) obrazek na obrazek w kolorach szarości (nie kombinuj, to nie jest grafika komputerowa, po prostu wpisz do każdej składowej średnią arytmetyczną pierwotnych wartości koloru).

14. `int height() const;`

    Po prostu zwraca wysokość obrazka. Może być zaimplementowana w pliku nagłówkowym `image.h`.

15. `int width() const;`

    Po prostu zwraca szerokość obrazka. Może być zaimplementowana w pliku nagłówkowym `image.h`.

Uwagi.

- Dobrze jest umieścić pliki `*.ppm` w tym samym katalogu, w którym generowany jest plik wykonywalny.
- Do zaokrąglania można użyć `std::round` (z nagłówka `<cmath>`)
- Rozwiązania nie muszą być zgodne ze sztuką grafiki komputerowej, wystarczy, że obrazki wynikowe będą z grubsza wyglądać tak, jak tego oczekujemy.
- Plik `CMakeLists.txt` dodałem dla tych, którzy wiedzą, co z nim zrobić. Pozostali mogą go zignorować.

Pytania kontrolne

1. Część funkcji składowych, np. `width()`, zadeklarowana jest z modyfikatorem `const`. Co to oznacza?
2. Do odczytywania plików zapisanych w formacie binarnym używam dodatkowego argumentu std::ios::binary. Dlaczego? Dobrze byłoby w tym miejscu opisać doświadczenia użytkowników systemu Windows, którzy w ramach eksperymentu pominęli ten argument.
3. Czym (pryncypialne) różnią się składowe `read/write` strumieni wejścia/wyjścia od poznanych dotąd operacji wykonywanych za pomocą `operator<<`?
4. Jaką rolę w formatach zapisu danych pełnią „liczby magiczne” – por. `https://en.wikipedia.org/wiki/List_of_file_signatures`, `https://en.wikipedia.org/wiki/Magic_number_(programming)#Format_indicator`
5. Na liście liczb magicznych (poprzedni punkt) odszukaj liczby magiczne dla formatu zip. Jakie inne formaty korzystają z tych samych liczb magicznych? Czy możesz zamienić rozszerzenie pliku zapisanego w formacie `docx`, `epub` lub `odt` na `zip` i traktować ten plik jako zwykły plik skompresowany w formacie `zip`? Do czego może się to przydać?
6. Dlaczego nazwy funkcji składowych w ich definicjach w pliku `*.cpp` poprzedza się nazwą klasy i operatorem `::`, np. `Image::save_as()`?
7. Moja implementacja składowej `read` sygnalizuje błędy instrukcją `throw`. Jaki jest domyślny efekt jej działania w programie?
8. Jak poradził(a/e)ś sobie z kompilacją programu składającego się z kilku plików źródłowych?
9. O co chodzi w zapisie literału napisowego `"\""`? Por.: `https://pl.wikipedia.org/wiki/Znak_modyfikacji`.
10. W deklaracji klasy `Image` posłużono się hermetyzacją danych. W jaki sposób?