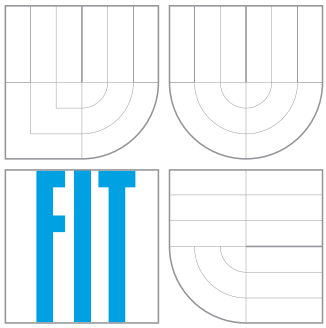


KRY

Projekt č. 2

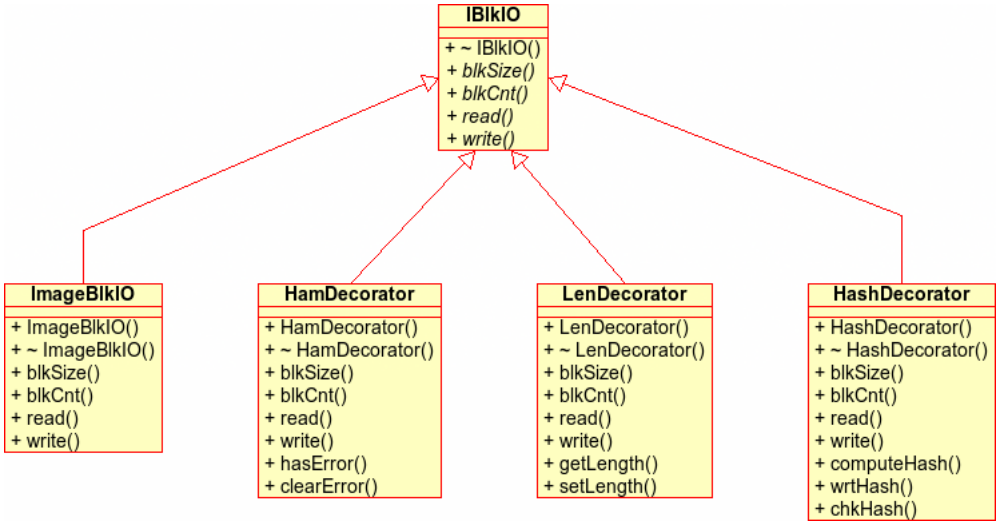
Kamil Dudka
xdudka00

1 Úvod

Úkolem bylo vytvořit program, který do určeného obrázku umí schovat text a tento text z obrázku později vytáhnout. Schovaný text měl být zabezpečený pomocí redundantního kódování a chráněný hashem. Navržený a implementovaný program byl rozšířen tak, aby do obrázku uměl schovat jakákoliv data. Pomocí programu lze tedy do obrázku ukládat text, ale kromě toho také jakýkoliv binární obsah, např. spustitelný soubor.

2 Objektový model

Na obrázku 1 je vidět objektový model programu. Na vrcholu hierarchie tříd je rozhraní `IBlkIO`, které funguje jako abstrakce blokového přístupu. Tohle rozhraní implementuje třída `ImageBlkIO`, která umí číst/zapisovat bloky dat z/do obrázku. Zbývající třídy představují tzv. *dekorátory* [1], které pracují s rozhraním `IBlkIO`. Tyto dekorátory implementují různé další funkciona-



Obrázek 1: Rozhraní `IBlkIO` a jeho implementace

lity, přičemž zachovávají původní rozhraní. Dekorátory jsou potom zřetězeny (podle obrázku) v pořadí zleva doprava. Třída `HamDecorator` zajišťuje kódování/dekódování bloků pomocí Hammingova kódu. Třída `LenDecorator` se stará o uložení/načtení délky přenášených dat – tím je umožněno přenášení binárních dat. Na konci řetězce dekorátorů je třída `HashDecorator`, která umí pracovat s hash funkcí md5. Při ukládání dat do obrázku je ulo-

žen také hash ukládaných dat. Při vytahování dat z obrázku je potom hash načten a porovnán s jeho vypočítanou hodnotou.

3 Implementace

V předchozí kapitole byl popsán návrh programu metodou shora dolů. Výsledkem návrhu jsou třídy, které jsou nezávislé na jejich implementaci. V této kapitole bude stručně popsána implementace jednotlivých tříd.

Třída `ImageBlkIO` implementuje obrázkový vstup/výstup pomocí knihovny `Qt3`. Tato knihovna zajistí načtení/uložení obrázku a přístup k jeho jednotlivým pixelům. Díky tomu program umí načíst obrázky různých formátů. Výsledný obrázek se ale ukládá vždy ve formátu PNG, který používá bezztrátovou kompresi. K samotnému schování dat se využívá LSB bit všech tří barevných složek každého pixelu.

Třída `HamDecorator` implementuje kódování/dekódování dat pomocí Hammingova kódu. K tomu využívá modul `hamming-0.3`, který je dostupný na http://michael.dipperstein.com/hamming/index.html a jeho zdrojové kódy jsou součástí archivu. Při sestavení programu se sestaví i tento modul a přilinkuje se k ostatním modulům během linkování.

Třída `LenDecorator` zajišťuje uložení údaje o délce přenášených dat spolu s daty. Její implementace je triviální a nepoužívá žádné další knihovny.

Třída `HashDecorator` implementuje výpočet md5 hashe pro přenášená data. Algoritmus výpočtu md5 hashe není implementován přímo, ale pomocí `gnulib` modulu `crypto/md5`, který je dostupný na http://www.gnu.org/software/gnulib/. Zdrojové kódy modulu jsou opět součástí archivu a při sestavení programu je zajištěn jejich překlad a linkování se zbytkem programu.

4 Návod k použití

Přeložený program se jmenuje `kry2`. Jméno programu se v programu nikde nevyskytuje – není tedy problém si vytvořit smysluplnější symlink, nebo spustitelný soubor přejmenovat. Program funguje ve třech režimech, které se volí tím, kolik se programu zadá parametrů na příkazové řádce:

- **žádný parametr** – Program vypíše nápovědu a skončí.
- **1 parametr** – Program očekává jméno souboru, ze kterého načte schovaná data a pošle je na standardní výstup. Pokud je při vytahování dat zjištěna opravitelná/neopravitelná chyba ECC, detekována neplatná

délka dat a/nebo odhalena neshoda hashe, jsou vypsána odpovídající chybové hlášky na standardní chybový výstup. Data (případně to, co z nich zbylo) jsou vypsána na standardní výstup i v těchto případech.

- **2 parametry** – Program očekává jméno vstupního souboru (obrázku v libovolném formátu) jako první parametr. Obrázek se načte a schovají se do něj data, která jsou načtena ze standardního vstupu. Potom je obrázek spolu s daty uložen do souboru, který je zadán jako druhý parametr. Výstupní obrázek je vždy ve formátu PNG bez ohledu na příponu výstupního souboru.

5 Výsledky

Výsledkem je program, který dokáže do obrázku schovat libovolná data. V této kapitole bude vypočtena velikost dat, která se dají do obrázku schovat. Množství dat samozřejmě záleží na konkrétní velikosti obrázku v pixelech. Díky objektovému modelu, který je založený na dekorátorech, bude tento výpočet velmi snadný. Každá třída implementující rozhraní `IBlkIO` musí definovat metody, které tuto informaci poskytují. Metoda `blkSize` vrací velikost bloku v bajtech a metoda `blkCnt` vrací maximální počet bloků, které lze do obrázku uložit.

Je zřejmé, že postupným vrstvením dekorátorů bude celkové množství dat, která se dají do obrázku uložit, klesat. Nejprve tedy vypočtěme velikost dat v případě použití samotné třídy `ImageBlkIO`. Třída pracuje vždy s bloky o velikosti 3 bajty. Každé 3 bajty jsou uloženy do LSB 8 pixelů obrázku. Pomocí trojčlenky můžeme snadno zjistit, že do obrázku o rozměrech w krát h, je možné uložit $w \cdot h \cdot \frac{3}{8}$ bajtů. Třída pracuje pouze s celými bloky – je tedy potřeba zaokrouhlovat vždy dolů.

Dále je v řetězci zapojen `HamDecorator`, který opět pracuje s 3-bajtovými bloky, snižuje však počet dostupných bloků na polovinu. To je dáno tím, že Hammingovo kódování je redundantní a každý `nibble` je kódován jako samostatný bajt. Ve výsledku to znamená, že počet uložitelných dat se také sníží na polovinu. Zaokrouhluje se opět směrem dolů (viz. výše).

Dekorátor `LenDecorator` využívá k uložení délky přenášených dat jeden blok, tedy 3 bajty. Tím je mimo jiné omezena délka přenášených dat na 2^{24} bajtů bez ohledu na velikost obrázku. Pokud by tento limit někoho omezoval, není problém v další verzi délku rozšířit na dva bloky a tím rozbít binární kompatibilitu se současným formátem. Protože je `LenDecorator` v řetězci až za `HamDecorator`, délka je také kódována redundantě. Ve výsledku je tedy množství uložitelných dat sníženo o 6 bajtů.

Dekorátor `HashDecorator` potřebuje do obrázku uložit hash dlouhý 16 bajtů. K dispozici má však 3-bajtové bloky musí tedy velikost zarovnat na 6 bloků (18 bajtů). Opět platí, že je v řetězci až za `HamDecorator` a tuto velikost je potřeba násobit dvěma. Množství uložitelných dat je tímto dekorátorem sníženo o 36 bajtů.

Dáme-li poznatky z analýzy jednotlivých dekorátorů dohromady, dostaneme vztah

$$MAX = w \cdot h \cdot \frac{3}{8} \cdot \frac{1}{2} - 1 - 18, \tag{1}$$

kde w je šířka obrázku a h je výška obrázku v pixelech. Při vyčíslování výrazu je potřeba pracovat v algebře celých čísel a při operaci dělení vždy zaokrouhlovat směrem dolů. To je způsobeno zarovnáním na celé bloky v průběhu zpracování dat (viz. výše).

6 Závěr

Jako školní projekt do předmětu KRY vznikl plnohodnotný a obecně využitelný program, který umí schovat text do obrázku a následně z něj tento text vytáhnout. Díky dobře zvládnuté analýze problému a efektivnímu využití již hotových komponent nebylo potřeba ztrácet čas implementací dříve implementovaného. Ušetřený čas jsem využil k implementaci jednoduchého avšak velmi přínosného rozšíření programu. Program dokáže kromě textu přenášet jakýkoliv binární obsah včetně spustitelných programů. Jako názorný příklad může sloužit obrázek 1, ve kterém je schovaný `tar-gz` archiv obsahující kompletní zdrojové kódy programu, jeho licenci a binární spustitelný soubor. Tento obrázek je součástí archivu. O jeho obsahu se můžete jednoduše přesvědčit spuštěním testovacího skriptu `check.sh`.

Reference

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley, 1997.